TORCONT v1 (2003)
Continuation Software for Quasi-Periodic 2-Tori

Frank Schilder,
Bristol Centre for Applied Nonlinear Mathematics,
Department of Engineering Mathematics,
University of Bristol, Bristol BS8 1TR, UK.

1. License

2. Download

This package will move to SourceForge:
http://sourceforge.net/projects/nlstools/

In the meantime, you may download it from
http://www.mathematik.tu-ilmenau.de/~fschild/nlsanalyzer/

3. Introduction

This continuation package consists of finder and continuer pairs of programs. It contains algorithms for computation (finder) and continuation (continuer) of

- fixed points (fpfind, fpcont),
- periodic solutions of autonomous and periodically forced systems (pofind, pocont) and
- quasiperiodic invariant 2-tori of autonomous and periodically forced systems (torfind, torcont; torfind4, torcont4).

The continuer use pseudo arclength continuation. pofind, pocont, torfind and torcont use a finite difference method of order 4. For torfind and torcont, error estimation is done by computing the difference of solutions obtained by methods of order 2 and 4. Therefore, the error is largely overestimated, because it is in fact an estimation for the solution of order 2.

torfind4 and torcont4 use a finite element method of order 1. There is no error estimation implemented yet. This programs give much nicer results than torfind and torcont, which is due to the fact that the FEM does not rely as much on smoothness. If you want to use the FEM within Maple, change the procedures torfind and torcont in contpack.mws accordingly and execute contpack.mws for updating the library contpack.m.

All algorithms are still experimental and not adaptive yet.

4. Installation

This is a stable pre-release of the continuation package TORCONT, written by Frank Schilder at the TU-Ilmenau, Germany.

Contents of directory nlsanalyzer-1.1 :

INSTALL	- this file
LICENSE	- copy of the FSF-GPL V2
Readme	- very brief overview of abilities
nlsanalyzer-1.1.tar.gz	- the complete source tree
examples.tar.gz	- examples of use (Maple worksheets)
nlsa-1.1-linux-bin.tar.gz	- binary distribution for LINUX

```
nlsa-1.1-sun-solaris-bin.tar.gz - binary distribution for SOLARIS
doc.tar.gz                     - documentation only
```

4.1. Documentation

```
You need gunzip and tar.

If you don't want to install the package, or just want to read trough the
documentation, then you may install the documentation only. Move the file
doc.tar.gz to a location of your choice. Change to this directory. Enter
the commands

gunzip doc.tar.gz
tar -xf doc.tar

This will create the directory doc which contains the documentation of
the Maple package contpack and of the examples in postscript format.
```

4.2. Installing the binary distribution

```
You need gunzip and tar.

The binary distribution consists of the files nlsa-1.1-linux-bin.tar.gz
for LINUX (on an Intel PC) and nlsa-1.1-sun-solaris-bin.tar.gz for SUN
SOLARIS (on a SUN Workstation), respectively, and the file examples.tar.gz
(both operating systems).

Move both files to a location of your choice. Change to this directory.
Enter the commands:

gunzip examples.tar.gz
gunzip nlsa-1.1-*-bin.tar.gz
tar -xf examples.tar
tar -xf nlsa-1.1-*-bin.tar

This will create the directories bin, maple and examples. These contain

bin      - scripts and binaries
maple    - maple package contpack
examples - four examples of use: kawa, lang, pnet, vdp

The binaries are linked against the following dynamic libraries. You can
use the binaries, if these libs are available on your system.

'ldd torcont' produced the following output:

LINUX:

libdl.so.2 => /lib/libdl.so.2 (0x40024000)
libstdc++-libc6.2-2.so.3 => /usr/lib/libstdc++-libc6.2-2.so.3 (0x40028000)
libm.so.6 => /lib/libm.so.6 (0x40075000)
libc.so.6 => /lib/libc.so.6 (0x40097000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)

SOLARIS:

libdl.so.1 =>  /usr/lib/libdl.so.1
libstdc++.so.2.10.0 =>  /usr/local/lib/libstdc++.so.2.10.0
libm.so.1 =>  /usr/lib/libm.so.1
libc.so.1 =>  /usr/lib/libc.so.1
/usr/platform/SUNW,Ultra-80/lib/libc_psr.so.1

You can use ldd to check, wether the libs are available or not. Simply
install the package and enter 'ldd torcont'. If there are problems, ldd
will most probably issue an error message.
```

4.3. Compiling the source distribution

```
Compiling the package is expensive. You will need 250MB RAM and
approximately 250MB free disc space.

The source code distribution consists of the files nlsanalyzer-1.1.tar.gz
and examples.tar.gz. It was compiled using the following compilers and
tools
```

program	version (SOLARIS)	version (LINUX)
gcc	2.95.3	2.95.3
g++	2.95.3	2.95.3
g77	2.95.3	2.95.3
flex	2.5.4	2.5.4
bison	1.35	1.28
gmake	3.79.1	3.79.1

You will need all this programs. You may try older or newer versions of flex, bison and gmake. You may try newer versions of gcc, g++, g77.

Move both files to a location of your choice. Change to this directory. Enter the commands:

```
gunzip examples.tar.gz
gunzip nlsanalyzer-1.1.tar.gz
tar -xf examples.tar
tar -xf nlsanalyzer-1.1.tar
```

This will create the directories examples and nlsanalyzer-1.1 . The directory examples contains the fully documented examples of use.

Change to nlsanalyzer-1.1 . Enter the command gmake (or make, if gmake is the default on your system). This will take a while. The package will compile with gmake only! In addition, the parser files are not lex and yacc compatible, so you will need flex and bison as listed above.

If the package is compiled successfully, then the subdirectory bin contains the executables, and maple the package contpack and its documentation. You should strip the executables, because the symbol information occupies very much of their size.

If gmake fails and you have the software versions listed above, or newer, please send me a log-file, which you obtain by the (csh-) commands:

```
gmake cleaner
gmake -i |& tee make.log
gzip make.log
```

Please send me the file make.log.gz by mail, subject: ‘make torcont’. Include into this mail the version info, which you obtain by the commands:

```
gcc -v
g++ -v
g77 -v
flex -V
bison -V
gmake -v
```

I will try to eliminate the errors and probably ask for further help or information.

Note: It is known that colpilation with newer versions of gcc fails. A new version is under development. **Do not send bug reports regarding this issue. They will not be dealt with.**

Documentation of the Maple-Package contpack.mws

General Remarks

We consider two kinds of equations, (periodically forced case):

$$\frac{\partial}{\partial t} x = \mathrm{f}(x, t)\,,$$

where *f* is 2 π-periodic with respect to *t*, and (autonomous case):

$$\frac{\partial}{\partial t} x = \mathrm{f}(x)\,,$$

with *x* element of R^n and *t* element of *R*. We seek periodic solutions

$$\mathrm{x}(t) = \mathrm{u}(\omega_1\, t)\,,$$

or quasiperiodic solutions

$$\mathrm{x}(t) = \mathrm{u}(\omega_1\, t, \omega_2\, t)\,.$$

In the periodically forced case, the frequency $\omega_1 = 1$ is equal to the forcing frequency. In addition, in the automomous case, it is possible to compute and continue fixed points.

The software computes and continues the torusfunction *u* (instead of the solution *x*), the unknown basic frequencies ω_1 and ω_2 and an estimation of the error. The latter is calculated as the L_2-norm of the difference of solutions of methods of order 2 and order 4 at the same mesh. The solution of order 4 is used, therefore the error is largely overestimated.

Executing this worksheet creates the file contpack.m, which is required if you want to use its functions.

This worksheet is best executed with the menu item Edit->Execute->Worksheet.
Please note the next paragraph before executing.

```
> restart:
  with(process):
```

Change the path to the location of the maple file ‘`contpack.mws`´ at your system.

```
> currentdir("/export/fschild/maple");
> 
```

Documentation

All functions of this package are described briefely in the following sections. Please refere to the Maple-example files also.

This documentation gives hints for using the command line programs without Maple. The expected format of the input and the parameter files are described in the sections of the preprocessing functions. Please read the script file ‘`demo`´, contained in each example’s subdirectory, on how to call the programs. Postprocessing can be done with e.g. gnuplot or Maple.

Continuation of invariant tori is usually done in two steps. At first, one has to compute a suitable initial solution (because one has usually crude approximations only). Then, one can do a parameter continuation, beginning at this start solution. Therefore, for all the considered kinds of solutions, there is a ‘finder´ and a ‘continuer´:

solution type	finder	continuer
fixed point	fpfind	fpcont
periodic sol.	pofint	pocont
quasiperiodic sol.	torfind	torcont

The ‘finder´ saves the solution found to a location, where it is expected by the corresponding ‘continuer´.

Preprocessing Procedures

ssf_name := **write_poss** (problem_name, run, solution, period, N)

(write periodic orbit start solution)

Creates a file containing the discretisation of the start solution in the format expected by pofind.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
solution - a function expecting a real number t as argument and returning a vector or list, containing the value of x(t); the period of this function must *be normalised to* 2*Pi
period - the (initial guess of the) period of the starting solution; for periodically forced systems, this period must be the exact value
N - the number of mesh-points

This function returns a string containing the unique name of the created file (*start solution file name*). This name must be used in the call to ‘create_pof_run´.

ssf_name := **write_tss** (problem_name, run, solution, period1, period2, N1, N2)

(write torus start solution)

Creates a file containing the discretisation of the start solution in the format expected by torfind.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
solution - a function expecting two real numbers th1 and th2 as arguments and returning a vector or list, containing the value of u(th1,th2); the two periods of this function must *be normalised to* 2*Pi
period1, period2 - the (initial guesses of the) periods of the starting solution; for periodically forced systems, period1 must be the exact value
N1, N2 - the number of discretisation-points on each axis

This function returns a string containing the unique name of the created file (*start solution f*

ile name). This name must be used in the call to ‘create_tf_run´.

create_ode (problem_name, ode, constants, params, options, ...)

Creates a shared library containing the right-hand-side of the ode.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
ode - the rhs of the ode, the function always expects the arguments x and t and returns a list of expressions resp. values
constants - a list of ‘name = value´-pairs for each constant of the ode
params - a list of ‘name = value´-pairs for each free parameter of the ode, an initial value must be provided
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
codegen = "codegen " - name des C-file generators, use always codegen
compiler = "gcc " - name and options of the C-compiler to be used
linker = "gcc -shared " - name and options of the linker to produce a shared object

create_fpf_run (problem_name, run, isol=x0, options, ...)

Creates the parameter file expected by fpfind.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
isol - the initial solution given as a list of values, if isol is not given, x0 is assumed to be the zero vector
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
ode.<parname> = <value> - sets the value of the free parameter <parname> to <value> (overwrites the default value)

nonlinear_solver.ItMX = 10 - max. number of Newton steps
nonlinear_solver.SubItMX = 8 - max. number of damping steps per Newton step
nonlinear_solver.TOL = 1.0e-4 - stopping criterion for the Newton iteration

create_fpc_run (problem_name, run, options, ...)

Creates the parameter file expected by fpcont.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
ode.<parname> = <value> - sets the value of the free parameter <parname> to <value> (overwrites the default value)

nonlinear_solver.ItMX = 10 - max. number of Newton steps
nonlinear_solver.SubItMX = 8 - max. number of damping steps per Newton step
nonlinear_solver.TOL = 1.0e-4 - stopping criterion for the Newton iteration

continuer.param = <parname> - set the primary continuation parameter
continuer.param_interval = [-1,1] - set the parameter interval
continuer.ItMX = 50 - set the maximum number of continuation steps in both directions
continuer.MaxDiff = 0.25 - set the maximum rel-abs-difference between predicted and corrected solution (for step size control)
continuer.Alpha = 7.0 - set the maximum angle between the tangent vectors of two solution points
continuer.h0 = 0.1 - set the initial continuation stepsize
continuer.h_max = 0.5 - set the maximal continuation stepsize
continuer.h_min = 0.01 - set the minimal continuation stepsize
continuer.LogFile - for printing debugging information of the continuer set LogFile to clog, this helps very much to tune the continuer

npr = 5 - print solution every npr steps

create_pof_run (problem_name, run, isol=file_name, options, ...)

Creates the parameter file expected by pofind.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
isol - the name of the file containing the initial solution (see write_poss)
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
ode.<parname> = <value> - sets the value of the free parameter <parname> to <value> (overwrites the default value)

discretisation_points = 20 - set the number of mesh points

linear_solver.LFil = 10 - set the number of values which are reserved for fill-in for each line of L and U computed by the ilu-preconditioner
linear_solver.Reserve = 20 - set the number of additional fill-in elements per line (if more than LFil elements violate the drop condition of the ilu-preconditioner
linear_solver.Restart = 15 - restart gmres after Restart iterations
linear_solver.ItMX = 150 - max. number of iterations of gmres, this is the global iteration index (counting subiterations)
linear_solver.DropTOL = 0.02 - dropping tolerance for the ilu-preconditioner
linear_solver.PermTOL = 1 - criterion for pivotisation, should always be set to 1
linear_solver.TOL = 1.0e-4 - the stopping criterion for gmres (relative residual)
linear_solver.LogFile = NULL - for printing debugging information of the linear solver set

linear_solver.LogFile = NULL - for printing debugging information of the linear solver set LogFile to clog, this helps very much to tune the linear solver

nonlinear_solver.ItMX = 10 - max. number of Newton steps
nonlinear_solver.SubItMX = 8 - max. number of damping steps per Newton step
nonlinear_solver.TOL = 1.0e-4 - stopping criterion for the Newton iteration

create_poc_run (problem_name, run, options, ...)

Creates the parameter file expected by pocont.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
ode.<parname> = <value> - sets the value of the free parameter <parname> to <value> (overwrites the default value)

discretisation_points = 20 - set the number of mesh points

linear_solver.LFil = 10 - set the number of values which are reserved for fill-in for each line of L and U computed by the ilu-preconditioner
linear_solver.Reserve = 20 - set the number of additional fill-in elements per line (if more than LFil elements violate the drop condition of the ilu-preconditioner
linear_solver.Restart = 15 - restart gmres after Restart iterations
linear_solver.ItMX = 150 - max. number of iterations of gmres, this is the global iteration index (counting subiterations)
linear_solver.DropTOL = 0.02 - dropping tolerance for the ilu-preconditioner
linear_solver.PermTOL = 1 - criterion for pivotisation, should always be set to 1
linear_solver.TOL = 1.0e-4 - the stopping criterion for gmres (relative residual)
linear_solver.LogFile = NULL - for printing debugging information of the linear solver set LogFile to clog, this helps very much to tune the linear solver

nonlinear_solver.ItMX = 10 - max. number of Newton steps
nonlinear_solver.SubItMX = 8 - max. number of damping steps per Newton step
nonlinear_solver.TOL = 1.0e-4 - stopping criterion for the Newton iteration

continuer.param = <parname> - set the primary continuation parameter
continuer.param_interval = [-1,1] - set the parameter interval
continuer.ItMX = 50 - set the maximum number of continuation steps in both directions
continuer.MaxDiff = 0.25 - set the maximum rel-abs-difference between predicted and corrected solution (for step size control)
continuer.Alpha = 7.0 - set the maximum angle between the tangent vectors of two solution points
continuer.h0 = 0.1 - set the initial continuation stepsize
continuer.h_max = 0.5 - set the maximal continuation stepsize
continuer.h_min = 0.01 - set the minimal continuation stepsize
continuer.LogFile - for printing debugging information of the continuer set LogFile to clog, this helps very much to tune the continuer

npr = 5 - print solution every npr steps

create_tf_run (problem_name, run, isol=file_name, options, ...)

Creates the parameter file expected by torfind.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
isol - the name of the file containing the initial solution (see write_tss)
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
ode.<parname> = <value> - sets the value of the free parameter <parname> to <value> (overwrites the default value)

discretisation_points1 = 20 - set the number of mesh points at the th1-axis
discretisation_points2 = 20 - set the number of mesh points at the th2-axis

linear_solver.LFil = 100 - set the number of values which are reserved for fill-in for each line of L and U computed by the ilu-preconditioner
linear_solver.Reserve = 200 - set the number of additional fill-in elements per line (if more than LFil elements violate the drop condition of the ilu-preconditioner
linear_solver.Restart = 35 - restart gmres after Restart iterations
linear_solver.ItMX = 350 - max. number of iterations of gmres, this is the global iteration index (counting subiterations)
linear_solver.DropTOL = 0.02 - dropping tolerance for the ilu-preconditioner
linear_solver.PermTOL = 1 - criterion for pivotisation, should always be set to 1
linear_solver.TOL = 1.0e-4 - the stopping criterion for gmres (relative residual)
linear_solver.LogFile = NULL - for printing debugging information of the linear solver set LogFile to clog, this helps very much to tune the linear solver

nonlinear_solver.ItMX = 10 - max. number of Newton steps
nonlinear_solver.SubItMX = 8 - max. number of damping steps per Newton step
nonlinear_solver.TOL = 1.0e-4 - stopping criterion for the Newton iteration

create_tc_run (problem_name, run, continuer.param=par_name, options, ...)

Creates the parameter file expected by pocont.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...
options - a sequence of ‘name = value´-pairs

Possible Options and Default Values:
ode.<parname> = <value> - sets the value of the free parameter <parname> to <value> (overwrites the default value)

discretisation_points1 = 20 - set the number of mesh points at the th1-axis
discretisation_points2 = 20 - set the number of mesh points at the th2-axis

linear_solver.LFil = 100 - set the number of values which are reserved for fill-in for each line of L and U computed by the ilu-preconditioner
linear_solver.Reserve = 200 - set the number of additional fill-in elements per line (if more than LFil elements violate the drop condition of the ilu-preconditioner
linear_solver.Restart = 35 - restart gmres after Restart iterations
linear_solver.ItMX = 350 - max. number of iterations of gmres, this is the global iteration index (counting subiterations)
linear_solver.DropTOL = 0.02 - dropping tolerance for the ilu-preconditioner
linear_solver.PermTOL = 1 - criterion for pivotisation, should always be set to 1
linear_solver.TOL = 1.0e-4 - the stopping criterion for gmres (relative residual)
linear_solver.LogFile = NULL - for printing debugging information of the linear solver set LogFile to clog, this helps very much to tune the linear solver

nonlinear_solver.ItMX = 10 - max. number of Newton steps
nonlinear_solver.SubItMX = 8 - max. number of damping steps per Newton step
nonlinear_solver.TOL = 1.0e-4 - stopping criterion for the Newton iteration

continuer.param = <parname> - set the primary continuation parameter
continuer.param_interval = [-1,1] - set the parameter interval
continuer.ItMX = 50 - set the maximum number of continuation steps in both directions
continuer.MaxDiff = 0.25 - set the maximum rel-abs-difference between predicted and corrected solution (for step size control)
continuer.Alpha = 7.0 - set the maximum angle between the tangent vectors of two solution points
continuer.h0 = 0.1 - set the initial continuation stepsize
continuer.h_max = 0.5 - set the maximal continuation stepsize
continuer.h_min = 0.01 - set the minimal continuation stepsize
continuer.LogFile - for printing debugging information of the continuer set LogFile to clog, this helps very much to tune the continuer

npr = 5 - print solution every npr steps

+

Procedures to call External Programs

fpfind (problem_name, run)

Runs the program fpfind and prints its output into the worksheet.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...

fpcont (problem_name, run)

Runs the program fpcont and prints its output into the worksheet. The run is timed.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...

pofind (problem_name, run)

Runs the program pofind and prints its output into the worksheet.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...

pocont (problem_name, run)

Runs the program pocont and prints its output into the worksheet. The run is timed.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...

torfind (problem_name, run)

Runs the program torfind and prints its output into the worksheet.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...

torcont (problem_name, run)

Runs the program torcont and prints its output into the worksheet. The run is timed.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - number of the run, e.g. 1, 2, ...

info := **odeinfo** (problem_name)

Runs the program odeinfo and returns two values in info:

info[1] - true or false wether or not the ode-system is autonomous
info[2] - dimension of the ode-system

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"

print_odeinfo (problem_name)

Runs the program odeinfo and prints its output into the worksheet.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"

+

Postprocessing Procedures

data := **read_bd** (problem_name, [run1, run2, ..., runN])

Reads the bifurcation diagrams of several runs into a list of lists.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
[run1, run2, ..., runN] - a list of the runs of interest

data := **select_bd_cols** (data, x, y)

Selects specified columns from bifurcation diagram data (for instance for plotting). Returns the values of the specified columns in a list of lists.

Parameters:

data - a list of lists containing the bifurcation diagram data
x, y - the two columns to be extracted

data := **read_po_data** (problem_name, run, orbit)

Read the data describing an periodic orbit into a list.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - the run the orbit was computed in
orbit - the number of the orbit within the run, see the output of pocont

data := **select_po_coords** (data, x, y, z)

Select the specified coordinates from a periodic orbit (for instance for plotting).

Parameters:

data - the data containing a complete orbit of a periodic orbit (dimension >3)

x,y,z - the columns to be extracted

data := **read_torus_data** (problem_name, run, orbit_num)

Read the data describing a 2-torus into a list.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - the run the torus was computed in
orbit - the number of the torus within the run, see the output of torcont

copy_periodic_solution (problem_name, run, num, new_run)

Copies a the periodic solution *num* of run *run* to the start solution of run *new_run*. Because pocont does not yet output orbits at specified parameter values, this function may not be of much use. If you want to restart from a solution at a different parameter value than these computed by pocont, consider using pofind with a start solution as close to the required parameter as possible. This is
equal to ‘copying and changing the parameter´. In future versions, this ‘detour´ should become unneseccary.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - the run the orbit was computed in
num - the number of the orbit within the run, see the output of pocont
new_run - the ‘target´ run for which the copied solution shall become the start solution

copy_torus_solution (problem_name, run, num, new_run)

Copies a the quasiperiodic solution *num* of run *run* to the start solution of run *new_run*. Because torcont does not yet output tori at specified parameter values, this function may not be of much use. If you want to restart from a solution at a different parameter value than these computed by torcont, consider using torfind with a start solution as close to the required parameter as possible. This is
equal to ‘copying and changing the parameter´. In future versions, this ‘detour´ should become unneseccary.

Parameters:

problem_name - an unique name of your problem as a string, e.g. "vdp"
run - the run the torus was computed in
num - the number of the torus within the run, see the output of torcont
new_run - the ‘target´ run for which the copied solution shall become the start solution

data := **select_torus_coords** (data, x, y, z)

Select the specified coordinates from a 2-torus (for instance for plotting).

Parameters:

data - the data containing a complete torus of a periodic orbit (phase-space dimension >3)
x,y,z - the columns to be extracted

− data := **cut1** (data, begin, end)

Cuts out a section of a 2-torus with respect to th1 from the mesh-point *begin* to the mesh-point *end*.

Parameters:

data - the data of a 2-torus
begin, end - the start- and end-segment

− data := **cut2** (data, begin, end)

Cuts out a section of a 2-torus with respect to th2 from the mesh-point *begin* to the mesh-point *end*.

Parameters:

data - the data of a 2-torus
begin, end - the start- and end-segment

− data := **section1** (data, idx)

Extracts a cross-section of a 2-torus for a fixed value of th1 at the mesh-point *idx*. This is the incariance curve of the stroboscopic map of the quasiperiodic solution with period T[1]=2*Pi/omega[1] locally defined near the torus.

Parameters:

data - the data of a 2-torus
idx - the index of the mesh-point

− data := **section2** (data, idx)

Extracts a cross-section of a 2-torus for a fixed value of th2 at the mesh-point *idx*. This is the incariance curve of the stroboscopic map of the quasiperiodic solution with period T[2]=2*Pi/omega[2] locally defined near the torus.

Parameters:

data - the data of a 2-torus
idx - the index of the mesh-point

+

```
> save write_poss, write_tss, xtd_system, create_ode,
       create_fpf_run, create_fpc_run,
```

```
        create_pof_run, create_poc_run,
        create_tf_run, create_tc_run,
        fpfind, fpcont, pofind, pocont, torfind, torcont,
        odeinfo, print_odeinfo,
        line_scan_format, read_bd, select_bd_cols,
        read_po_data, select_po_coords,
        read_torus_data, copy_periodic_solution,
        copy_torus_solution, select_torus_coords,
        cut1, cut2, section1, section2,
        "contpack.m";
> 
```

The Example of C. Hayashi, T. Yoshinaga and H. Kawakami

This example was given by T. Yoshinaga and H. Kawakamiand shows a cascade of torus doubling bifurcations which lead to a strange attractor. This is a nice test example because no (numerical) obserbable resonances occour. The system is given by the equations

$$\frac{\partial}{\partial t} x = y$$

$$\frac{\partial}{\partial t} y = -k_1\, y - \frac{1\,(x^2 + 3\,z^2)\,x}{8} + B\cos(t)$$

$$\frac{\partial}{\partial t} z = -\frac{1\,k_2\,(3\,x^2 + z^2)\,z}{8} + B_0$$

with the parameter values $k_2 = .5\text{e-}1$, $B = .22$, $B_0 = .3\text{e-}1$ and $k = [.4\text{e-}1 \,..\, .15]$.

Attention! Before starting the computations, please set the paths in the ‘read´ and the ‘currentdir´ commands correctly. Ignore the error message issued by ‘mkdir´.

```
> restart;
  with(plots):
  with(process):
  read "/export/fschild/maple/contpack.m":
  currentdir("/export/fschild/examples/kawa");
  mkdir("data"):
  currentdir();
Warning, the name changecoords has been redefined

Error, (in mkdir) directory exists and is not empty
```

"/export/fschild/examples/kawa"

[>
[>
[>
[>

Definition of the System and Creation of the Shared Object

The constants and parameters are defined as a list of ‘name = value´ pairs.
The system is defined as a function taking a list and a number as arguments and returning a list of expressions.

```
> Constants := [k2 = 0.05, B  = 0.22, B0 = 0.03]:
  Params    := [k1 = 0.09 ]:

  Kawa := (x,t) -> [
    x[2],
    -k1*x[2]-1/8*(x[1]^2+3*x[3]^2)*x[1]+B*cos(t),
    -1/8*k2*(3*x[1]^2+x[3]^2)*x[3]+B0
  ]:
```

Create a shared object by calling codegen and then compiling and linking.
The compiler options used are for Solaris. You may need different options.

```
> create_ode("kawa", Kawa, Constants, Params,
```

```
    compiler = "gcc -fPIC",
    linker   = "gcc -fPIC -shared"
  );
codegen < kawa.ode > kawa.c ... OK
gcc -fPIC -c -o kawa.o kawa.c ... OK
gcc -fPIC -shared -o kawa.so kawa.o ... OK
```

Definition of the Start Solutions

Define initial approximations to periodic (psol), quasiperiodic (tsol) and the doubled quasiperiodic (dtsol) solutions. This functions (i.e. the coefficients) were obtained by Fourier analysis of orbits, which were the result of numerical simulation.
Unfortunately, there are no branch-switching-algorithms available yet, so we need this initial guesses.

```
> psol := (t) -> [
    +1.70583E-03+5.17355E-01*sin(t)-8.05828E-01*cos(t),
    +1.47861E-04+7.99765E-01*sin(t)+5.20006E-01*cos(t),
    +1.32976E+00+1.91934E-04*sin(t)+3.91183E-05*cos(t)
  ]:

  tsol := (t,th) -> [
    +1.70583E-03        +5.17355E-01*sin(t)
  -8.05828E-01*cos(t)

  -1.08938E-04*sin(th)+6.34324E-01*sin(t)*sin(th)-4.00134E-02*c
  os(t)*sin(th)

  +3.36430E-03*cos(th)-8.63900E-02*sin(t)*cos(th)+4.43208E-01*c
  os(t)*cos(th),

    +1.47861E-04        +7.99765E-01*sin(t)
  +5.20006E-01*cos(t)

  -1.23093E-03*sin(th)+6.06000E-02*sin(t)*sin(th)+5.95267E-01*c
  os(t)*sin(th)

  +4.01392E-04*cos(th)-3.69231E-01*sin(t)*cos(th)-9.74376E-02*c
  os(t)*cos(th),

    +1.32976E+00        +1.91934E-04*sin(t)
  +3.91183E-05*cos(t)

  +6.18952E-02*sin(th)-6.28914E-05*sin(t)*sin(th)-8.69171E-05*c
  os(t)*sin(th)

  +9.68285E-02*cos(th)+3.91103E-04*sin(t)*cos(th)+9.41343E-05*c
  os(t)*cos(th)
  ]:

  dtsol := (t,th) -> [
  -6.3293e-05+4.7468e-01*sin(t)-8.1639e-01*cos(t)-6.1526e-04*si
  n(1.0*th)-9.9029e-02*sin(t)*sin(1.0*th)+8.2113e-03*cos(t)*sin
```

```
(1.0*th)+3.1443e-04*cos(1.0*th)+3.9425e-02*sin(t)*cos(1.0*th)
-8.6842e-03*cos(t)*cos(1.0*th)-7.1716e-04*sin(2.0*th)-3.0896e
-01*sin(t)*sin(2.0*th)+5.1457e-01*cos(t)*sin(2.0*th)+1.0355e-
03*cos(2.0*th)-5.9266e-01*sin(t)*cos(2.0*th)-1.0655e-01*cos(t
)*cos(2.0*th)+2.9828e-05*sin(3.0*th)-6.5673e-02*sin(t)*sin(3.
0*th)+7.2169e-02*cos(t)*sin(3.0*th)+1.7579e-03*cos(3.0*th)-7.
1634e-02*sin(t)*cos(3.0*th)-7.8122e-02*cos(t)*cos(3.0*th)+1.1
177e-03*sin(4.0*th)-8.6009e-02*sin(t)*sin(4.0*th)+1.6180e-01*
cos(t)*sin(4.0*th)+1.6492e-03*cos(4.0*th)-1.4964e-01*sin(t)*c
os(4.0*th)-9.3327e-02*cos(t)*cos(4.0*th)+1.7377e-03*sin(5.0*t
h)-4.2339e-02*sin(t)*sin(5.0*th)+1.1509e-02*cos(t)*sin(5.0*th
)+9.3014e-04*cos(5.0*th)-1.6307e-02*sin(t)*cos(5.0*th)-4.4285
e-02*cos(t)*cos(5.0*th),

+1.7772e-04+8.1657e-01*sin(t)+4.7682e-01*cos(t)-7.2020e-04*si
n(1.0*th)-1.1031e-02*sin(t)*sin(1.0*th)-9.9793e-02*cos(t)*sin
(1.0*th)+7.2372e-04*cos(1.0*th)+3.3203e-03*sin(t)*cos(1.0*th)
+4.2824e-02*cos(t)*cos(1.0*th)-9.0885e-04*sin(2.0*th)-4.5432e
-01*sin(t)*sin(2.0*th)-3.0015e-01*cos(t)*sin(2.0*th)+1.2109e-
03*cos(2.0*th)+7.2740e-02*sin(t)*cos(2.0*th)-5.3331e-01*cos(t
)*cos(2.0*th)-8.1945e-04*sin(3.0*th)-6.1829e-02*sin(t)*sin(3.
0*th)-5.4459e-02*cos(t)*sin(3.0*th)+1.8827e-03*cos(3.0*th)+6.
6952e-02*sin(t)*cos(3.0*th)-5.5852e-02*cos(t)*cos(3.0*th)-2.2
792e-04*sin(4.0*th)-1.3274e-01*sin(t)*sin(4.0*th)-6.9366e-02*
cos(t)*sin(4.0*th)+2.1520e-03*cos(4.0*th)+7.0248e-02*sin(t)*c
os(4.0*th)-1.1002e-01*cos(t)*cos(4.0*th)+2.5432e-04*sin(5.0*t
h)-8.3238e-03*sin(t)*sin(5.0*th)-2.9773e-02*cos(t)*sin(5.0*th
)+2.1189e-03*cos(5.0*th)+3.2202e-02*sin(t)*cos(5.0*th)-9.3574
e-03*cos(t)*cos(5.0*th),

+1.3092e+00-1.0911e-04*sin(t)-5.1342e-05*cos(t)-1.7796e-02*si
n(1.0*th)-1.0779e-04*sin(t)*sin(1.0*th)-1.5922e-05*cos(t)*sin
(1.0*th)-4.1941e-02*cos(1.0*th)-1.6124e-04*sin(t)*cos(1.0*th)
-9.3525e-05*cos(t)*cos(1.0*th)+5.9965e-02*sin(2.0*th)-1.2945e
-04*sin(t)*sin(2.0*th)-3.2461e-05*cos(t)*sin(2.0*th)-9.9713e-
02*cos(2.0*th)-3.9658e-05*sin(t)*cos(2.0*th)-8.3610e-05*cos(t
)*cos(2.0*th)-1.8230e-03*sin(3.0*th)-4.0360e-05*sin(t)*sin(3.
0*th)-3.5066e-05*cos(t)*sin(3.0*th)-6.2331e-03*cos(3.0*th)+7.
7939e-05*sin(t)*cos(3.0*th)-6.0643e-05*cos(t)*cos(3.0*th)-2.3
175e-03*sin(4.0*th)+1.0078e-04*sin(t)*sin(4.0*th)-2.0976e-05*
cos(t)*sin(4.0*th)-9.0188e-03*cos(4.0*th)+9.4243e-05*sin(t)*c
os(4.0*th)-3.5578e-05*cos(t)*cos(4.0*th)-1.4481e-03*sin(5.0*t
h)+2.3374e-04*sin(t)*sin(5.0*th)+1.8662e-05*cos(t)*sin(5.0*th
)-7.2119e-04*cos(5.0*th)-8.0794e-07*sin(t)*cos(5.0*th)-2.7773
e-05*cos(t)*cos(5.0*th)
]:
```

Write discretisations of the initial solutions to unique files.
This takes some time, so do not call this functions if not necessary.

Note that the first period is always equal to the forcing period. For the second period we give initial guesses (60.4 and 118). The computations are done on a 20x40-mesh (single torus) and a 20x80-mash (doubled torus).

```
> # Args:    name    run  isol   T1    T2      N1  N2

  write_poss("kawa", 1,   psol,  2*Pi,         20    );

  write_tss ("kawa", 2,   tsol,  2*Pi,  60.4, 20, 40);
  write_tss ("kawa", 3,   dtsol, 2*Pi, 118,   20, 80);
                           "kawa_po1.dat"
                          "kawa_qpo2.dat"
                          "kawa_qpo3.dat"
```

[>

Run 1: Continuation of a Periodic Orbit

At first, we continue the periodic orbit, from which the invariant torus emerges by a Hopf bifurcation.

This run demonstrates, how to
- run pofind and pocont.
- create 3D-plots of computed periodic orbits.

Create the parameter file expected by ‘pofind´ with settings
- problem name: "kawa"
- run: 1
- initial solution file: "kawa_po1.dat"

and then compute the initial periodic orbit.

```
> create_pof_run("kawa", 1, isol = "kawa_po1.dat");
  pofind("kawa", 1);
Iterat    D mpfung                  Normen                    Rechenzeit
  I SI       gamma       ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  7.3236e+00  2.2614e-01  0.0000e+00      0      0      0
  1  1  1.0000e+00  7.7135e+00  2.8230e-02  5.8477e-01    0.0    0.0    0.0
  2  1  1.0000e+00  7.6929e+00  7.9791e-04  1.2612e-01    0.0    0.0    0.0
  3  1  1.0000e+00  7.6918e+00  6.6770e-06  4.8005e-03    0.0    0.0    0.0
  4  1  1.0000e+00  7.6918e+00  2.3129e-07  1.9336e-05    0.0    0.0    0.0

solution written to file 'data/kawa.1/po0.dat'

*** checking for memory leaks ... no leaks.
```

Create the parameter file expected by ‘pocont´ with settings
- problem name: "kawa"
- run: 1
- continuation parameter: k1
- continuation interval: [0.04, 0.15]
- print every npr steps, npr: 1

and then continue the periodic solution. The solution written to "data/kawa.1/po0.dat" by pofind is used as initial solution.

```
> create_poc_run("kawa", 1, continuer.param = k1,
    continuer.param_interval = [0.04, 0.15],
    npr = 1
  );
  pocont("kawa", 1);
  STEP         PAR         ||x|| data file
     0   9.000e-02   1.7199e+00 data/kawa.1/po0.dat
     1   9.438e-02   1.7195e+00 data/kawa.1/po1.dat
```

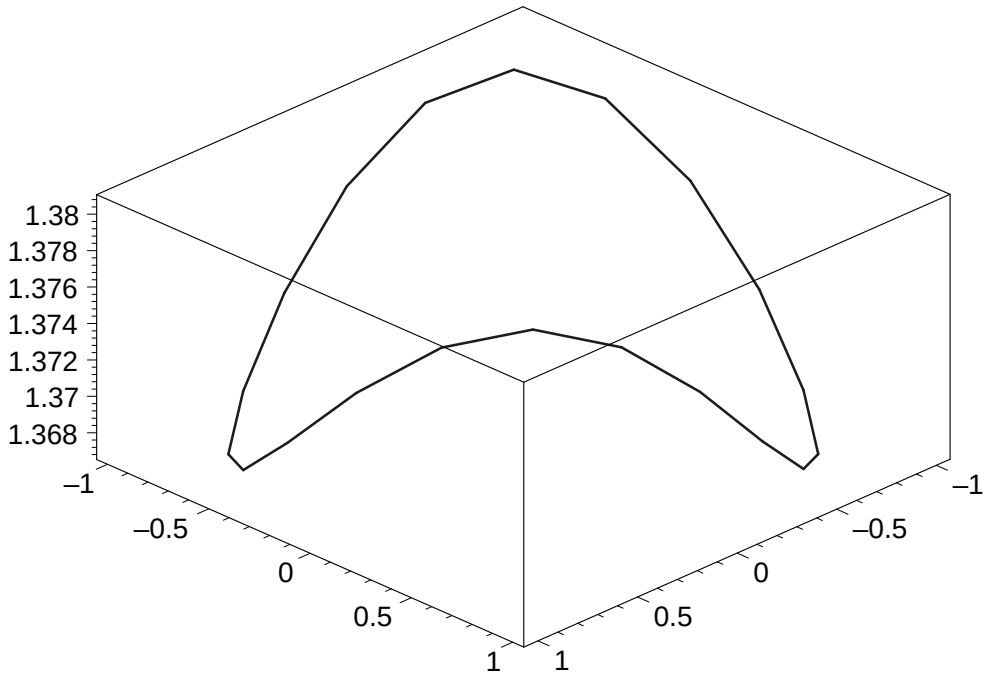
```
     2   1.027e-01   1.7186e+00 data/kawa.1/po2.dat
     3   1.187e-01   1.7167e+00 data/kawa.1/po3.dat
     4   1.361e-01   1.7144e+00 data/kawa.1/po4.dat
     5   1.535e-01   1.7121e+00 data/kawa.1/po5.dat

  STEP         PAR         ||x|| data file
     0   9.000e-02   1.7199e+00 data/kawa.1/po0.dat
     1   8.563e-02   1.7204e+00 data/kawa.1/po6.dat
     2   7.735e-02   1.7212e+00 data/kawa.1/po7.dat
     3   6.166e-02   1.7225e+00 data/kawa.1/po8.dat
     4   4.442e-02   1.7236e+00 data/kawa.1/po9.dat
     5   2.677e-02   1.7244e+00 data/kawa.1/po10.dat

bifurcation diagramm written to file: 'data/kawa.1/bd.dat'
output written to file:               'data/kawa.1/pocont.log'

*** checking for memory leaks ... no leaks.

real        0.6
user        0.4
sys         0.0
```

Read the computed data of the periodic orbit into data1. Here orbit 0 of run 1 is choosen. Then select the columns 2,3,4 from the data, which contain the projection onto the (x,y,z)-subspace.

The data of a periodic orbit in R^n at N mesh-points has the following structure (note: t0 = tN):

```
[
  [ t0, x1(t0), x2(t0), ..., xn(t0) ],
    .
    .
    .
  [ tN, x1(tN), x2(tN), ..., xn(tN) ]
]
```

```
> data1:=read_po_data("kawa",1,0):
  data :=select_po_coords(data1, 2,3,4):
> spacecurve({data}, thickness=3, color=blue, axes=boxed);
```



[>

Run 2: Continuation of a Quasiperiodic Orbit / Invariant 2-Torus

Secondly, we continue the invariant torus, emerging from the periodic orbit by a Hopf bifurcation.

This run demonstrates, how to
- run torfind and torcont.
- create 3d-plots of computed tori.

Create the parameter file expected by ‘torfind´ with settings
- problem name: "kawa"
- run: 2
- initial solution file: "kawa_qpo2.dat"
- mesh: 20x40
- min. number of nonzero entries per row of L and U: 50
- max. number of additional nonzero elements is rows*reserve, reserve: 100
- dropping tolerance for the ilu-factorisation: 0.02
- don’t print debugging information, LogFile: NULL

and then compute the initial torus.

Note: You should always provide enough space for nonzeros generated by ilu (by setting LFil and Reserve to resonable high values), because the linear systems are very hard to solve. The iteration methods are very sensitive to dropping too much. This is valid for all of the computations.

```
> create_tf_run ("kawa", 2, isol="kawa_qpo2.dat",
```

```
    discretisation_points1 = 20,
    discretisation_points2 = 40,
    linear_solver.LFil = 50,
    linear_solver.Reserve = 100,
    linear_solver.DropTOL = 0.02,
    linear_solver.LogFile = NULL # clog
  );
  torfind("kawa", 2);
Iterat    D mpfung                  Normen                  Rechenzeit
  I SI       gamma       ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  4.8714e+01  1.6097e+00  0.0000e+00      0      0      0
  1  1  1.0000e+00  4.8993e+01  1.5010e-01  4.4725e+00    0.1    0.8    3.9
  2  1  1.0000e+00  4.8789e+01  5.5154e-03  8.3250e-01    0.2    1.6    5.1
  3  1  1.0000e+00  4.8781e+01  2.8667e-04  2.6383e-02    0.4    2.4    6.2
  4  1  1.0000e+00  4.8781e+01  4.8317e-06  3.8652e-03    0.5    3.1    7.3
  5  1  1.0000e+00  4.8781e+01  5.0101e-07  7.9926e-05    0.6    3.9    8.4

period T2 = 6.149773868198037973 8e+01
solution written to file 'data/kawa.2/qpo0.dat'

*** checking for memory leaks ... no leaks.
```

Create the parameter file expected by 'torcont´ with settings
- problem name: "kawa"
- run: 2
- continuation parameter: k1
- mesh: 20x40
- dropping tolerance: 0.01
- max. norm of relative residual for gmres (TOL): 1.0e-8
- don't print dedugging information, LogFile: NULL
- continuation interval: [0.04, 0.12]
- max. continuation step size: 1.0

and then continue the invariant torus.

```
> create_tc_run ("kawa", 2, continuer.param=k1,
    discretisation_points1 = 20,
    discretisation_points2 = 40,
    linear_solver.DropTOL = 0.01,
    linear_solver.TOL = 1.0e-8,
    linear_solver.LogFile = NULL, # clog,
    continuer.param_interval = [0.04, 0.12],
    continuer.h_max = 1.0
  );
  torcont("kawa", 2);
  STEP         PAR        ||x||          TOL    Period T2 data file
     0   9.000e-02   1.7247e+00   1.0136e-01   6.1498e+01 data/kawa.2/qpo0.d
at
     5   9.902e-02   1.7221e+00   1.1373e-01   6.3415e+01 data/kawa.2/qpo1.d
at
    10   1.128e-01   1.7183e+00   1.4946e-01   6.6012e+01 data/kawa.2/qpo2.d
at
    15   1.197e-01   1.7167e+00   1.9703e-01   6.7101e+01 data/kawa.2/qpo3.d
at
    16   1.204e-01   1.7166e+00   1.9810e-01   6.7206e+01 data/kawa.2/qpo4.d
at

  STEP         PAR        ||x||          TOL    Period T2 data file
     0   9.000e-02   1.7247e+00   1.0136e-01   6.1498e+01 data/kawa.2/qpo0.d
at
     5   7.953e-02   1.7277e+00   9.1782e-02   5.9108e+01 data/kawa.2/qpo5.d
at
    10   5.752e-02   1.7339e+00   8.1960e-02   5.3408e+01 data/kawa.2/qpo6.d
at
    15   3.828e-02   1.7396e+00   7.9678e-02   4.7345e+01 data/kawa.2/qpo7.d
```

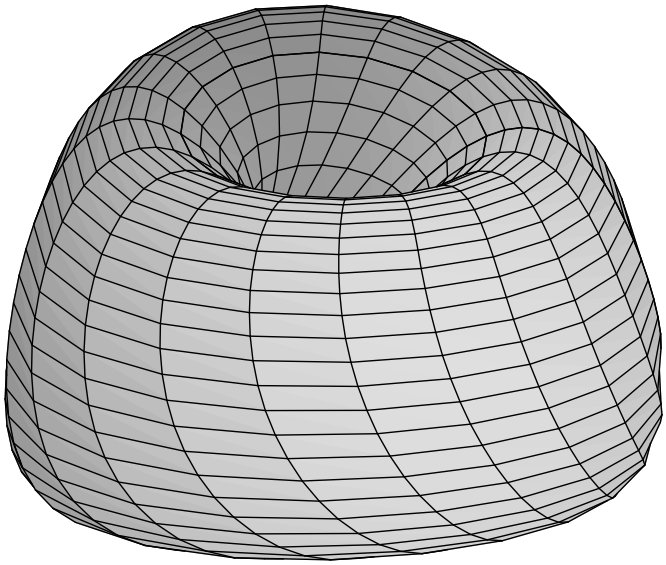
```
    15   3.828e-02   1.7396e+00   7.9678e-02   4.7345e+01 data/kawa.2/qpo7.d
at

bifurcation diagramm written to file: 'data/kawa.2/bd.dat'
output written to file:               'data/kawa.2/torcont.log'

*** checking for memory leaks ... no leaks.

real     4:33.4
user     4:32.8
sys         0.1
```

Read the computed data of the invariant torus into data1. Here torus 0 of run 2 is choosen. Then select the columns 3,4,5 from the data, which contain the projection onto the (x,y,z)-subspace.

The data of a 2-torus in R^n at a N1xN2 mesh has the following structure (note: th1_0 = th1_N1, th2_0 = th2_N2):

```
[
  [
    [ th1_0 , th2_0 , x1(th1_0 , th2_0 ), x2(th1_0 , th2_0 ), ..., xn(th1_0 , th2_0 ) ],
      .
      .
      .
    [ th1_N1, th2_0 , x1(th1_N1, th2_0 ), x2(th1_N1, th2_0 ), ..., xn(th1_N1, th2_0 ) ]
  ],
  [
    [ th1_0 , th2_1 , x1(th1_0 , th2_1 ), x2(th1_0 , th2_1 ), ..., xn(th1_0 , th2_1 ) ],
      .
      .
      .
    [ th1_N1, th2_1 , x1(th1_N1, th2_1 ), x2(th1_N1, th2_1 ), ..., xn(th1_N1, th2_1 ) ]
  ],
  .
  .
  .
  [
    [ th1_0 , th2_N2, x1(th1_0 , th2_N2), x2(th1_0 , th2_N2), ..., xn(th1_0 , th2_N2) ],
      .
      .
      .
    [ th1_N1, th2_N2, x1(th1_N1, th2_N2), x2(th1_N1, th2_N2), ..., xn(th1_N1, th2_N2) ]
  ]
]
```

```
> data1:=read_torus_data("kawa", 2, 0):
  data :=select_torus_coords(data1, 3,4,5):
> surfdata({data});
```



[>

Run 3: Continuation of the Doubled Quasiperiodic Orbit / Invariant 2-Torus

Thirdly, we continue the doubled invariant torus, emerging from the ‘simple´ torus by a torus-doubling bifurcation in the second basic frequency.

This run demonstrates, how to
- run torfind and torcont.
- cut out sections of tori.
- extract cross-sections (corresponding to invariant circles of local stroboscopic maps).
- create 3d-plots of computed tori together with space-curves.

Create the parameter file expected by ‘torfind´ with settings
- problem name: "kawa"
- run: 3
- initial solution file: "kawa_qpo3.dat"
- owerwrite the value of k1 to: 0.0775
- mesh: 20x80 (doubled with respect to th2)
- LFil and Reserve: 100, 200
- dropping tolerance for ilu: 0.01
- don’t print debugging information, LogFile: NULL

and then compute the initial torus.

```
> create_tf_run ("kawa", 3, isol="kawa_qpo3.dat",
    ode.k1 = 0.0775,
    discretisation_points1 = 20,
    discretisation_points2 = 80,
    linear_solver.LFil = 100,
    linear_solver.Reserve = 200,
    linear_solver.DropTOL = 0.01,
    linear_solver.LogFile = NULL # clog
  );
  torfind("kawa", 3);
Iterat    D mpfung                  Normen                  Rechenzeit
  I SI       gamma       ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  6.9101e+01  2.1539e+00  0.0000e+00      0      0      0
  1  1  1.0000e+00  6.9168e+01  2.9498e-02  2.1636e+00    0.3    1.3   13.7
  2  1  1.0000e+00  6.9134e+01  1.6709e-03  5.0092e-01    0.5    2.5   17.5
  3  1  1.0000e+00  6.9132e+01  1.1526e-04  6.1077e-02    0.8    3.8   21.2
  4  1  1.0000e+00  6.9132e+01  9.5981e-06  2.3312e-03    1.1    5.0   24.9
  5  1  1.0000e+00  6.9132e+01  3.8010e-07  2.4931e-04    1.3    6.2   28.7
  6  1  1.0000e+00  6.9132e+01  4.2433e-08  1.2618e-05    1.5    7.4   32.5

period T2 = 1.1719963027479205664e+02
solution written to file 'data/kawa.3/qpo0.dat'

*** checking for memory leaks ... no leaks.
```

Create the parameter file expected by ‘torcont´ with settings
- problem name: "kawa"
- run: 3
- continuation parameter: k1
- owerwrite the value of k1 to: 0.0775
- mesh: 20x80
- LFil and Reserve: 300, 300
- max. number of iterations of gmres (ItMX): 700
- max. value of rel. residual of gmres (TOL): 1.0e-8
- continuation interval: [0.04, 0.08]
- max. continuation step size: 1.0

and then continue the invariant torus.

In the output: Note the jump in the tolerance for k[1]=[0.05877 .. 0.05725], which is due to a number of weak resonances.

```
> create_tc_run ("kawa", 3, continuer.param=k1,
    ode.k1 = 0.0775,
    discretisation_points1 = 20,
    discretisation_points2 = 80,
    linear_solver.LFil = 300,
    linear_solver.Reserve = 300,
    linear_solver.ItMX = 700,
    linear_solver.TOL = 1.0e-8,
    continuer.param_interval = [0.04, 0.08],
    continuer.h_max = 1.0
  );
  torcont("kawa", 3);
  STEP         PAR         ||x||          TOL    Period T2 data file
     0   7.750e-02   1.7283e+00   1.3375e-01   1.1720e+02 data/kawa.3/qpo0.d
at
     5   7.917e-02   1.7278e+00   1.1748e-01   1.1804e+02 data/kawa.3/qpo1.d
at
     8   8.006e-02   1.7276e+00   9.2680e-02   1.1847e+02 data/kawa.3/qpo2.d
```

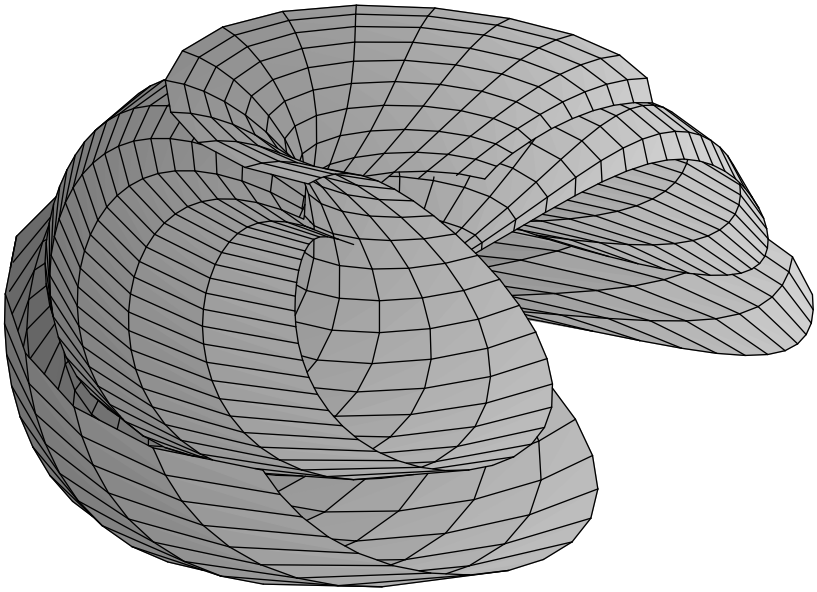
```
     8   8.006e-02   1.7276e+00   9.2680e-02   1.1847e+02 data/kawa.3/qpo2.d
at

  STEP         PAR         ||x||          TOL    Period T2 data file
     0   7.750e-02   1.7283e+00   1.3375e-01   1.1720e+02 data/kawa.3/qpo0.d
at
     5   7.505e-02   1.7290e+00   1.3710e-01   1.1584e+02 data/kawa.3/qpo3.d
at
    10   6.826e-02   1.7313e+00   1.3451e-01   1.1107e+02 data/kawa.3/qpo4.d
at
    15   6.219e-02   1.7345e+00   1.9847e-01   1.0459e+02 data/kawa.3/qpo5.d
at
    20   5.877e-02   1.7386e+00   1.1273e+00   9.8277e+01 data/kawa.3/qpo6.d
at
    25   5.589e-02   1.7439e+00   6.0845e-01   9.3032e+01 data/kawa.3/qpo7.d
at
    30   5.065e-02   1.7491e+00   3.0263e-01   8.9128e+01 data/kawa.3/qpo8.d
at
    35   4.508e-02   1.7520e+00   2.5987e-01   8.6736e+01 data/kawa.3/qpo9.d
at
    39   3.962e-02   1.7535e+00   2.6043e-01   8.4890e+01 data/kawa.3/qpo10.
dat

bifurcation diagramm written to file: 'data/kawa.3/bd.dat'
output written to file:               'data/kawa.3/torcont.log'

*** checking for memory leaks ... no leaks.

real    24:53.4
user    24:51.7
sys         0.5
```

Read the computed data of the invariant torus into data1. Here torus 0 of run 3 is choosen. Then select the columns 3,4,5 from the data, which contain the projection onto the (x,y,z)-subspace.

```
> data1:=read_torus_data("kawa", 3, 0):
  data2:=select_torus_coords(data1, 3,4,5):
```

Different from the previous plots, we want to cut out some segments for a better view. In this example we cut out parts in both direction to show, that this is (successively) possible. We obtain a nice view into the doubled torus.

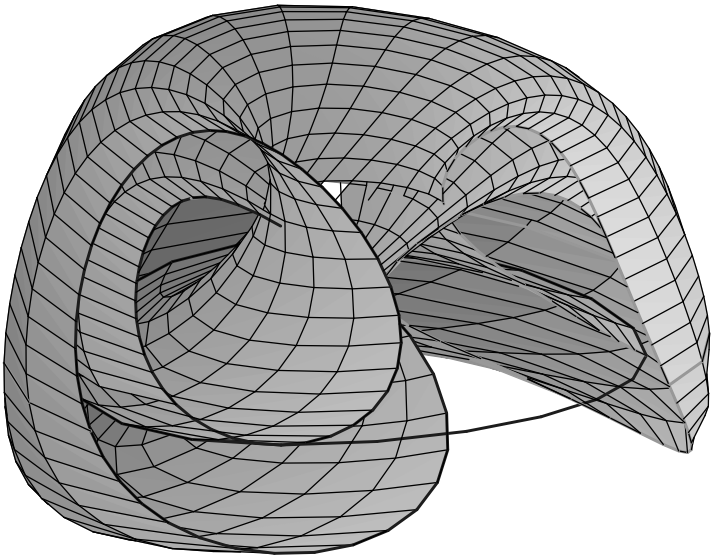
```
> data3:=cut1(data2,  4, 10):
  data4:=cut2(data3, 56, 70):
  gr1:=surfdata({data4}):
  display(gr1);
```



The following demonstrates, how to obtain cross sections (corresponding to invariance curves of local stroboscopic maps), and how to draw these together with the torus. Note that we create cross-sections of both, the full (blue) and the dissected (coral) torus.

```
> data5:=section1(data2, 10):
  #data6:=section1(data2,  4):
  data7:=section2(data2, 36):
  #data8:=section2(data2, 31):

  #data5:=section1(data4, 1):
  data6:=section1(data4,  15):
  #data7:=section2(data4, 1):
  data8:=section2(data4,  36):

  gr2:=spacecurve(data5, thickness=3, color=blue):
  gr3:=spacecurve(data6, thickness=3, color=coral):
  gr4:=spacecurve(data7, thickness=3, color=blue):
  gr5:=spacecurve(data8, thickness=3, color=coral):
```

```
> display([gr1, gr2, gr3, gr4, gr5]);
```



[>

Bifurcation Diagram

At last, we draw the bifurcation diagram.

The bifurcation diagrams contain the following columns in each row:

fpcont: `PAR ||x|| <components of fixed point>`

pocont (non-autonomous): `PAR ||x||`

pocont (autonomous): `PAR ||x|| T`

torcont (non-autonomous): `PAR ||x|| ERR T2`

torcont (autonomous): `PAR ||x|| ERR T1 T2`

where means:

`PAR` - value of parameter
`||x||` - normalised L2-norm of x
`T` - period (periodic orbit)
`ERR` - estimated error of a quasiperiodic solution
`T1` - first basic period (quasiperiodic orbit)

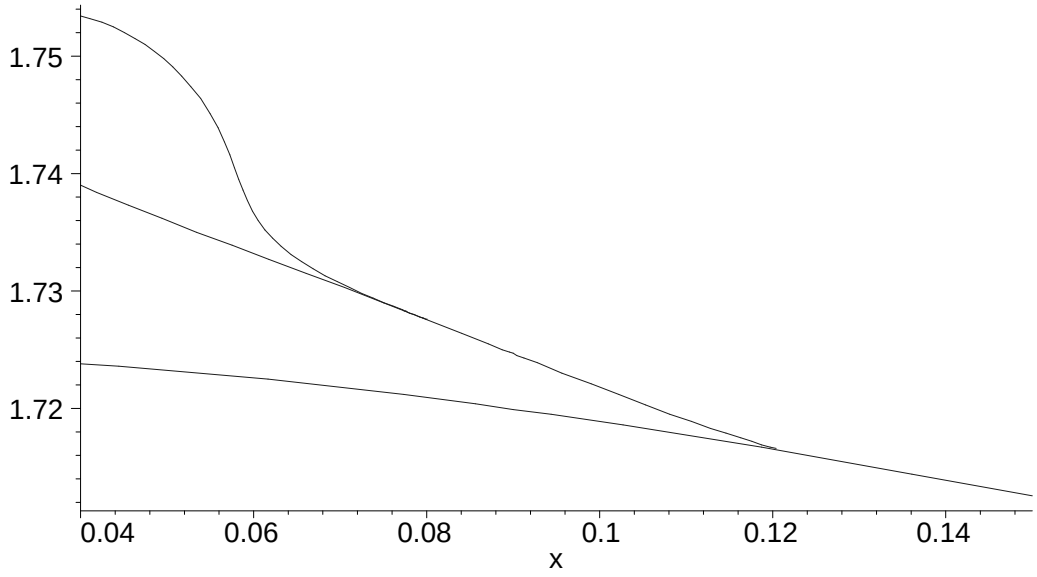
`T2` - second basic period (quasiperiodic orbit)

The data structure of a bifurcation diagram is a list of such rows (which are also lists). By reading in multiple bifurcation diagrams, you get a list of such lists (which may have different formats).

Read in the bifurcation diagrams of the different runs.

Note, that the bifurcation diagrams of different runs contain different numbers of columns, and the same data may appear in columns depending on the problem type. But the first two columns always contain the parameter and the norm of the solution. Therefore, we can select the columns 1 and 2 from all these data sets (otherwise, we would have to do this for each bifurcation diagram separately).

```
> data1 := read_bd("kawa", [1,2,3]):
  data1 := select_bd_cols(data1, 1, 2):
  plot(data1, x=0.04..0.15, color=blue);
```



[>

[>

The Nonlinear Parametrically forced Network

The parametrically forced network is described by the equation

$$\left(\frac{\partial^2}{\partial t^2}x\right)+\alpha\left(\frac{\partial}{\partial t}x\right)^3-\beta\left(\frac{\partial}{\partial t}x\right)+(1+B\sin(2\,t))\,x=0$$

and is of interest to electrical engineers. It is investigated in dependence of epsilon and the remaining parameters are given by

$$B=.1$$
$$\alpha=\varepsilon-B$$
$$\beta=\frac{\varepsilon}{2}-B$$

The system is constructed to posess a so called subharmonic response solution with halved frequency (frequence divider) for epsilon=B. Here, we investigate the system for epsilon=[1.5 .. 6.5].

Attention! Before starting the computations, please set the paths in the ‘read´ and the ‘currentdir´ commands correctly. Ignore the error message issued by ‘mkdir´.

```
> restart;
  with(plots):
  with(process):
  read "/export/fschild/maple/contpack.m":
  currentdir("/export/fschild/examples/pnet");
  mkdir("data"):
  currentdir();
Warning, the name changecoords has been redefined

Error, (in mkdir) directory exists and is not empty
```

"/export/fschild/examples/pnet"

[>
[>
[>
[>

Definition of the System and Creation of the Shared Object

The constants and parameters are defined as a list of ‘name = value´ pairs.
The system is defined as a function taking a list and a number as arguments and returning a list of expressions.

```
> Constants := [alpha=epsilon-B, beta=epsilon/2-B]:
  Params    := [epsilon=3.0, B=0.1]:

  PNet := (x,t) -> [
    x[2],
    -alpha*x[2]^3+beta*x[2]-(1+B*sin(2*t))*x[1]
  ];
```

$$PNet := (x,\,t)\rightarrow[x_2,\,-\alpha\,x_2^{\,3}+\beta\,x_2-(1+B\sin(2\,t))\,x_1]$$

Create a shared object by calling codegen and then compiling and linking.
The compiler options used are for Solaris. You may need different options.

```
> create_ode("pnet", PNet, Constants, Params,
    compiler = "gcc -fPIC",
    linker   = "gcc -fPIC -shared"
  );
codegen < pnet.ode > pnet.c ... OK
gcc -fPIC -c -o pnet.o pnet.c ... OK
gcc -fPIC -shared -o pnet.so pnet.o ... OK
```

Definition of the Start Solutions

Define initial approximations to the quasiperiodic solution.
This function is a guess.

This torus function(s) must always be 2*Pi-periodic in each argument.

```
> isol := (t,th) -> [sin(th), 0.8*cos(th)]:
```

Write the initial solution to disk on a 30x100 mesh. The first period is Pi, the second is an initial guess. The name returned is used in the first call to torfind (run 1). Here we raise Digits to full double precision to obtain a most exact mesh in th1, which is fixed through all computations. This is usually not necessary, but shown in this example. This is especially *not* neccessary for autonomous systems,
because the periods are determined by the extended system with the required accuracy!

Parameters:
- problem name: "pnet"
- run: 1
- name of the function calculating initial solution values: isol
- T1: Pi
- T2: 6.96
- mesh: 30x100

```
> Digits:=19:
  write_tss("pnet", 1, isol, Pi, 6.96, 30, 100);
  Digits:=10:
```

"pnet_qpo1.dat"

Run 1: Continuation of Quasiperiodic Orbits: eps=[1.5..6.5]

This run demonstrates, how to
- run torfind and torcont.
- create 3D-plots and animations.

Create the parameter file expected by 'torfind' with settings
- problem name: "pnet"
- run: 1
- initial solution file: "pnet_qpo1.dat"
- mesh: 30x100
- LFil and Reserve (see also ex. kawa): 250, 250
- use a Krylov subspace of at most Restart dimensions, Restart: 35
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.01

- max. value of rel residual of gmres (TOL): 1.0e-8
- don't print debugging information of iterative solver and ilu, LogFile: NULL

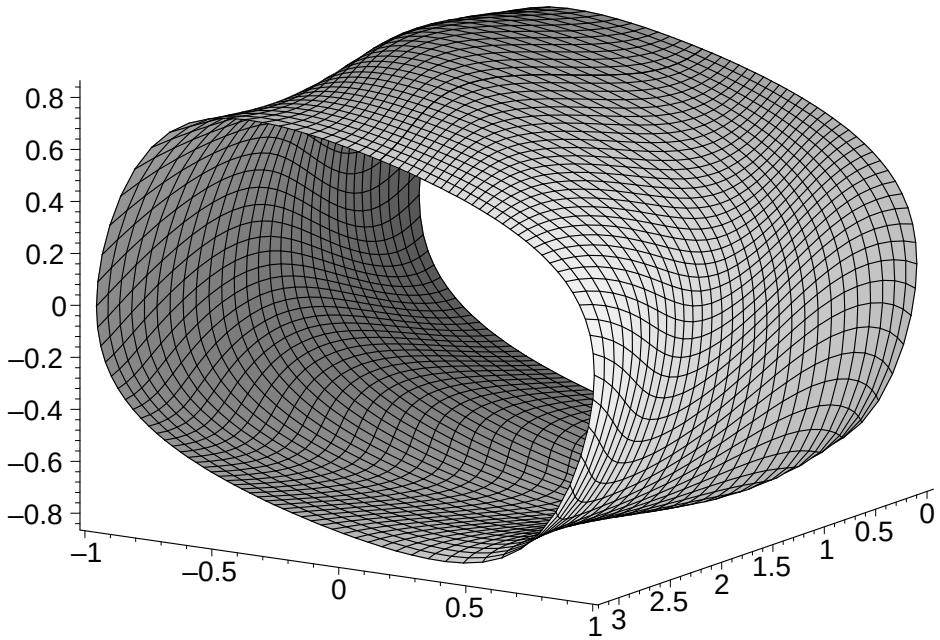
and run torfind.

```
> create_tf_run ("pnet", 1, isol="pnet_qpo1.dat",
    discretisation_points1 = 30,
    discretisation_points2 = 100,
    linear_solver.LFil = 250,
    linear_solver.Reserve = 250,
    linear_solver.Restart = 35,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.01,
    linear_solver.TOL = 1.0e-8,
    linear_solver.LogFile = NULL # clog
  );
  torfind("pnet", 1);
```

```
Iterat    D mpfung                  Normen                    Rechenzeit
  I SI       gamma        ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  4.9607e+01  1.8597e+01  0.0000e+00      0      0      0
  1  1  1.0000e+00  4.8752e+01  5.9276e+00  8.6662e+00    0.4    1.9   24.7
  2  1  1.0000e+00  4.7434e+01  6.5696e-01  2.9055e+00    0.8    3.6   35.1
  3  1  1.0000e+00  4.7311e+01  9.6584e-03  2.9172e-01    1.1    5.4   44.5
  4  1  1.0000e+00  4.7309e+01  1.8311e-06  3.6620e-03    1.5    7.1   53.9
  5  1  1.0000e+00  4.7309e+01  1.8487e-10  1.1100e-06    1.8    8.9   63.7

period T2 = 6.9584467149239035422e+00
solution written to file 'data/pnet.1/qpo0.dat'

*** checking for memory leaks ... no leaks.
```

Create s plot of the computed torus in 'phase space´ (t, x, $\frac{\partial}{\partial t}x$). Because t is not plotted periodically, the torus appears not closed in the plot.
See the example "kawa" for the format of the data structure.

```
> data1:=read_torus_data("pnet", 1, 0):
  data2 :=select_torus_coords(data1, 1,3,4):
  surfdata(data2);
```

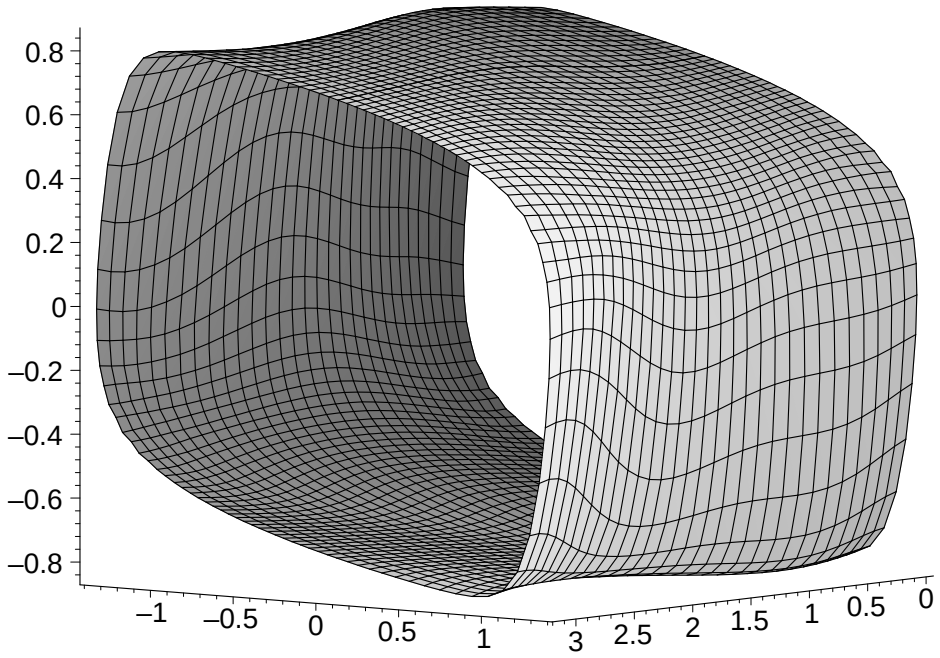


Create the parameter file expected by 'torcont' with settings
- problem name: "pnet"
- run: 1
- continuation parameter: epsilon
- mesh: 30x100
- LFil and Reserve (see also ex. kawa): 100, 200
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.005
- max. value of rel residual of gmres (TOL): 1.0e-8
- don't print debugging information of iterative solver and ilu, LogFile: NULL
- continuation interval: [1.5, 7.0]
- max. number of continuation steps: 150
- print every npr steps, npr: 5

and run torcont.

```
> create_tc_run ("pnet", 1, continuer.param=epsilon,
    discretisation_points1 = 30,
    discretisation_points2 = 100,
    linear_solver.LFil = 100,
    linear_solver.Reserve = 200,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.005,
    linear_solver.TOL = 1.0e-8,
    linear_solver.LogFile = NULL, # clog,
    continuer.param_interval = [1.5, 7.0],
    continuer.ItMX = 150,
    npr = 5
  );
```

```
  torcont("pnet", 1);
  STEP         PAR          ||x||          TOL    Period T2 data file
     0   3.000e+00   8.6358e-01   5.7013e-03   6.9584e+00 data/pnet.1/qpo0.d
at
     5   3.355e+00   8.8125e-01   4.0472e-03   7.1356e+00 data/pnet.1/qpo1.d
at
    10   3.934e+00   9.1147e-01   2.7886e-03   7.4471e+00 data/pnet.1/qpo2.d
at
    15   4.531e+00   9.4411e-01   2.3816e-03   7.7914e+00 data/pnet.1/qpo3.d
at
    20   4.932e+00   9.6678e-01   2.5076e-03   8.0341e+00 data/pnet.1/qpo4.d
at
    25   5.478e+00   9.9835e-01   1.2610e-02   8.3755e+00 data/pnet.1/qpo5.d
at
    30   5.507e+00   1.0000e+00   1.7242e-02   8.3937e+00 data/pnet.1/qpo6.d
at
    35   5.515e+00   1.0005e+00   2.1305e-02   8.3985e+00 data/pnet.1/qpo7.d
at
    40   5.520e+00   1.0008e+00   2.3145e-02   8.4016e+00 data/pnet.1/qpo8.d
at
    45   5.524e+00   1.0010e+00   2.4256e-02   8.4042e+00 data/pnet.1/qpo9.d
at
    50   5.528e+00   1.0012e+00   2.4240e-02   8.4066e+00 data/pnet.1/qpo10.
dat
    55   5.532e+00   1.0014e+00   2.2975e-02   8.4093e+00 data/pnet.1/qpo11.
dat
    60   5.537e+00   1.0018e+00   2.0943e-02   8.4129e+00 data/pnet.1/qpo12.
dat
    65   5.546e+00   1.0023e+00   1.8557e-02   8.4190e+00 data/pnet.1/qpo13.
dat
    70   5.567e+00   1.0036e+00   1.7132e-02   8.4325e+00 data/pnet.1/qpo14.
dat
    75   5.602e+00   1.0057e+00   1.7816e-02   8.4552e+00 data/pnet.1/qpo15.
dat
    80   5.722e+00   1.0128e+00   2.3569e-02   8.5325e+00 data/pnet.1/qpo16.
dat
    85   6.281e+00   1.0463e+00   1.1788e-02   8.9001e+00 data/pnet.1/qpo17.
dat
    90   6.745e+00   1.0748e+00   6.4120e-02   9.2135e+00 data/pnet.1/qpo18.
dat
    95   6.866e+00   1.0823e+00   6.5148e-02   9.2947e+00 data/pnet.1/qpo19.
dat
   100   6.941e+00   1.0859e+00   3.6025e-01   9.3314e+00 data/pnet.1/qpo20.
dat
   105   6.975e+00   1.0868e+00   7.7580e-01   9.3359e+00 data/pnet.1/qpo21.
dat
   110   7.005e+00   1.0876e+00   4.1035e-01   9.3406e+00 data/pnet.1/qpo22.
dat

  STEP         PAR          ||x||          TOL    Period T2 data file
     0   3.000e+00   8.6358e-01   5.7013e-03   6.9584e+00 data/pnet.1/qpo0.d
at
     5   2.687e+00   8.4858e-01   1.0881e-02   6.8123e+00 data/pnet.1/qpo23.
dat
    10   2.385e+00   8.3450e-01   1.9077e-02   6.6801e+00 data/pnet.1/qpo24.
dat
damped_newton: no convergence

bifurcation diagramm written to file: 'data/pnet.1/bd.dat'
output written to file:               'data/pnet.1/torcont.log'

*** checking for memory leaks ... no leaks.

real  1:19:53.5
user  1:19:37.3
sys         1.5
> data1:=read_torus_data("pnet", 1, 17):
  data2 :=select_torus_coords(data1, 1,3,4):
  surfdata(data2);
```



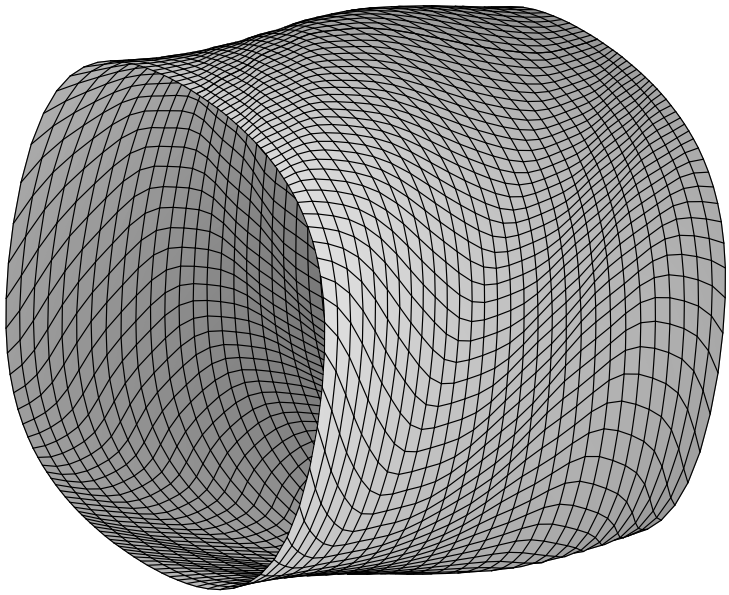
Here, we create an animation of the evolution of the torus under changes of the parameter epsilon.
At first, we read in all solutions given in a list (nums). The for-loop creates a sequence of plot-structures.

```
> nums:=[24, 23, $0..22]:
  an:=NULL:
  for i from 1 by 1 to nops(nums) do
    printf("reading graph %d\n", nums[i]);
    data1:=read_torus_data("pnet", 1, nums[i]):
    data2 :=select_torus_coords(data1, 1,3,4):
    an:=an, surfdata({data2}):
  od:
reading graph 24
reading graph 23
reading graph 0
reading graph 1
reading graph 2
reading graph 3
reading graph 4
reading graph 5
reading graph 6
reading graph 7
reading graph 8
reading graph 9
reading graph 10
reading graph 11
reading graph 12
reading graph 13
reading graph 14
reading graph 15
reading graph 16
reading graph 17
reading graph 18
reading graph 19
reading graph 20
reading graph 21
reading graph 22
```

[>

In this animation, you can see some tori, which are not smooth. This is due to week and strong resonances and occours visibly araund epsilon=5.524 and epsilon=7.05. The strong resonance (at 7.05) can be observed as a phase-lock, the ‘detected´ weak resonance may by due to the relatively crude approximation and does not appear so clearly for finer meshes.

```
> display([an], insequence=true);
```



[>

[>

[>

Bifurcation Diagram

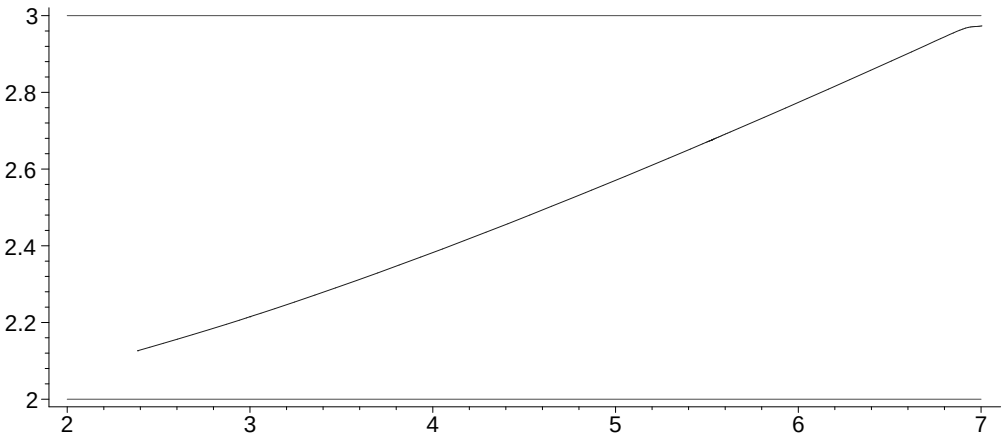
For growing epsilon, the system runs into a strong (1:3) resonance around epsilon=7.05. For detecting resonances, one can compute the ratio of the two periods (which gives the rotation number or its inverse) and do a ‘rational analysis´. This simply means, that we look, if this ratio crosses or approaches rational numbers which belong to strong resonances (for example 1:1, 1:2, ..., 1:4).

A plot of the inverse rotation number (for the stroboscopic map with period Pi). It is clearly to see, that the system runs into an 1:2 resonance as epsilon tends to zero, and into an 1:3 resonance as epsilon tends to values greater than 7. Of course, between these two strong resonances, there are some more or less weak resonances, see therefore also the next plot.

We read the bifurcation diagram into data1. data1 then contains two lists of data, one for the forward continuation and one for the backward continuation. From both lists we extract the parameter and the second basic period. By dividing the second period by Pi (the first basic period), we obtain the inverse rotation number. These is plotted (blue) together with the lines of the 1:2 and the 1:3 resonances (red) over the parameter epsilon.

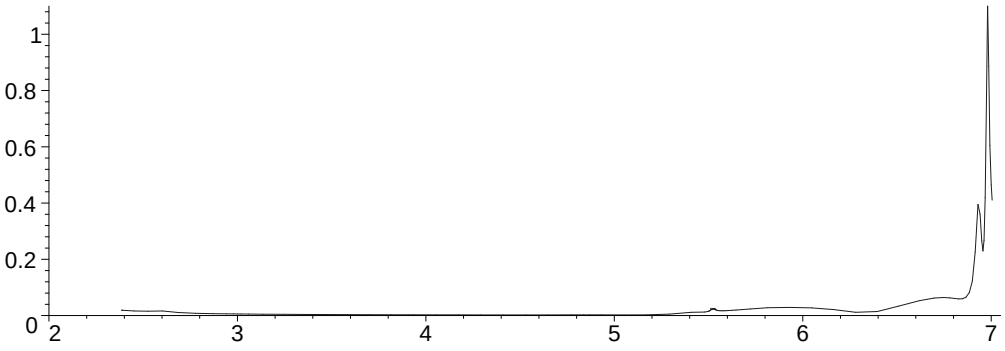
See example "kawa" for the format of the bifurcation diagram data structure.

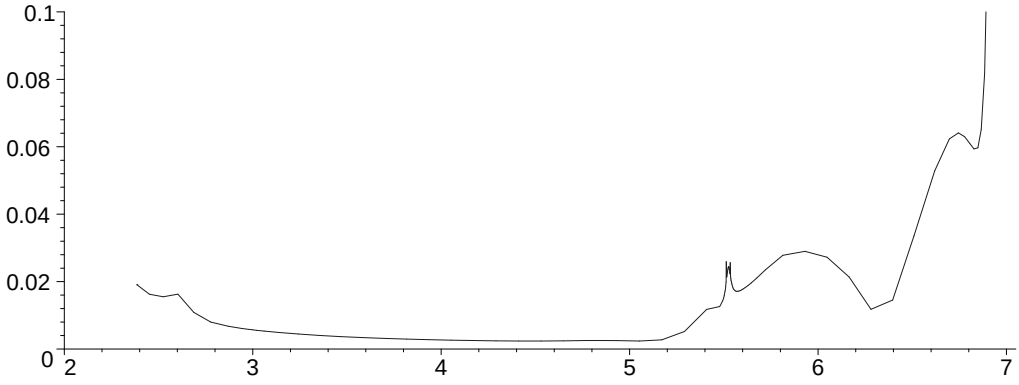
```
> data1 := read_bd("pnet", [1]):
  data2 := select_bd_cols(data1, 1, 4):
  data3 := NULL:
  for i from 1 by 1 to nops(data2[1]) do
    data3 := [data2[1][i][1], evalf(data2[1][i][2]/Pi)], data3:
  od:
  for i from 1 by 1 to nops(data2[2]) do
    data3 := data3, [data2[2][i][1], evalf(data2[2][i][2]/Pi)]:
  od:
  pl := plot([[data3]], color=blue):
  pl := pl, plot([[[2,2], [7,2]], [[2,3], [7,3]]], color=red):
  display(pl);
```



A plot of the estimated error of the torus solution. The error is plottet over epsilon. Areas with larger error values indicate, where ‘ilands´ of weak resonances may reside. But numerically, this is very hard to verify. At strong resonances, our method breaks down, which is shown by the peak of the error near epsilon=7.

```
> data2 := select_bd_cols(data1, 1, 3):
  plot(data2, 2..7.05, 0..1.1, color=blue);
  plot(data2, 2..7.05, 0..0.1, color=blue);
```




0.1
0.08
0.06
0.04
0.02
0
2
3
4
5
6
7

The Example of W.F. Langford

W.F. Langford considered a dynamical system of Hydrodynamics under the following transformations:

1. Reduction to the 3-dimensional center manifold and
2. Transformation into the Poincare-Birkhoff-Normal Form.

Omitting higher order terms he derived the system

$$\frac{\partial}{\partial t} x = (z - .7)\, x - \omega\, y$$

$$\frac{\partial}{\partial t} y = \omega\, x + (z - .7)\, y$$

$$\frac{\partial}{\partial t} z = .6 + z - \frac{z^3}{3} - (x^2 + y^2)\,(1 + \rho\, z) + \varepsilon\, z\, x^3$$

which is considered here for the parameter values epsilon=[0..0.1], rho=[0.15..0.7] and omega=3.5. For epsilon=0 and rho>0.615446... the system posesses a periodic orbit, which for rho=0.615446... undergoes a Hopf bifurcation, and an attractive invariant 2-torus emerges for rho>0.615446... For smaller values of rho, a Shilnikov-type attractor is born in a global bifurcation involving the torus (see run 1). Values of epsilon>0 lead to strong resonances (see run 2).

Attention! Before starting the computations, please set the paths in the ‘read´ and the ‘currentdir´ commands correctly. Ignore the error message issued by ‘mkdir´.

```
> restart;
  with(plots):
  with(process):
  read "/export/fschild/maple/contpack.m":
  currentdir("/export/fschild/examples/lang");
  mkdir("data");
  currentdir();
Warning, the name changecoords has been redefined

Error, (in mkdir) directory exists and is not empty
```

"/export/fschild/examples/lang"

[>
[>
[>
[>

Definition of the System and Creation of the Shared Object

The constants and parameters are defined as a list of ‘name = value´ pairs.
The system is defined as a function taking a list and a number as arguments and returning a list of expressions.

```
> Constants := []:
  Params    := [epsilon = 0, rho = 0.55, omega = 3.5]:

  LANG := (x,t) -> [
```

```
    (x[3]-0.7)*x[1]-omega*x[2],
    omega*x[1]+(x[3]-0.7)*x[2],

  0.6+x[3]-1/3*x[3]^3-(x[1]^2+x[2]^2)*(1+rho*x[3])+epsilon*x[3]
  *x[1]^3
  ]:
```

Create a shared object by calling codegen and then compiling and linking.
The compiler options used are for Solaris. You may need different options.

```
> create_ode("lang", LANG, Constants, Params,
    compiler = "gcc -fPIC",
    linker   = "gcc -fPIC -shared"
  );
codegen < lang.ode > lang.c ... OK
gcc -fPIC -c -o lang.o lang.c ... OK
gcc -fPIC -shared -o lang.so lang.o ... OK
```

[>

Definition of the Start Solutions

Define initial approximations to the quasiperiodic solution.
This function is a guess.

This torus function(s) must always be 2*Pi-periodic in each argument.

```
> isol := (t,th) -> [
    (0.9+0.3*cos(th))*cos(t),
    (0.9+0.3*cos(th))*sin(t),
    0.7-0.5*sin(th)
  ]:
```

Write the initial solution to disk on a 40x40 mesh. The first period is exactly known (from analysis of the system), the second is an initial guess. The name returned is used in the first call to torfind (run 1).

Parameters:
- problem name: "lang"
- run: 1
- name of the function calculating initial solution values: isol
- T1: 2*Pi/3.5
- T2: 4.31
- mesh: 40x40

```
> write_tss("lang", 1, isol, 2*Pi/3.5, 4.31, 40, 40);
```

"lang_qpo1.dat"

[>

Run 1: Continuation of Quasiperiodic Orbits: rho=[0.15 .. 0.7]

This run demonstrates, how to
- run torfind and torcont.
- obtain and interpret debugging information of the linear solver.
- create 3D-plots and animations.

Create the parameter file expected by 'torfind' with settings
- problem name: "lang"
- run: 1
- initial solution file: "lang_qpo1.dat"
- mesh: 40x40
- LFil and Reserve (see also ex. kawa): 200, 250
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.0025
- max. value of rel residual of gmres (TOL): 1.0e-12
- print debugging information of iterative solver and ilu, LogFile: clog

and run torfind.

```
> create_tf_run ("lang", 1, isol="lang_qpo1.dat",
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 200,
    linear_solver.Reserve = 250,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.0025,
    linear_solver.TOL = 1.0e-12,
    linear_solver.LogFile = clog
  );
  torfind("lang", 1);
Iterat    D mpfung                  Normen                    Rechenzeit
  I SI       gamma       ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  4.8645e+01  4.6590e+00  0.0000e+00      0      0      0
PGMRES: ilutp: time: 7.6, nnz(A): 72004, nnz(LU): 1292166, max. nnz: 3081301
PGMRES: pgmres(32): time: 4.2
  1  1  1.0000e+00  4.7493e+01  1.0849e-01  1.9569e+00    0.3    3.1   31.4
PGMRES: ilutp: time: 7.2, nnz(A): 72004, nnz(LU): 1282896, max. nnz: 3081301
PGMRES: pgmres(24): time: 3.2
  2  1  1.0000e+00  4.7473e+01  2.5990e-04  6.5099e-02    0.5    6.1   42.2
PGMRES: ilutp: time: 7.2, nnz(A): 72004, nnz(LU): 1283517, max. nnz: 3081301
PGMRES: pgmres(27): time: 3.6
  3  1  1.0000e+00  4.7473e+01  7.5179e-05  4.5610e-02    0.8    9.1   53.4
PGMRES: ilutp: time: 7.2, nnz(A): 72004, nnz(LU): 1283536, max. nnz: 3081301
PGMRES: pgmres(26): time: 3.4
  4  1  1.0000e+00  4.7473e+01  1.5044e-08  6.4515e-04    1.0   12.1   64.5
PGMRES: ilutp: time: 7.2, nnz(A): 72004, nnz(LU): 1283510, max. nnz: 3081301
PGMRES: pgmres(26): time: 3.5
  5  1  1.0000e+00  4.7473e+01  2.3314e-12  5.1449e-08    1.3   15.2   75.7

period T1 = 1.7951594779940944768e+00
period T2 = 4.3115354542660266901e+00
solution written to file 'data/lang.1/qpo0.dat'
estimating error ...
PGMRES: ilutp: time: 2.05, nnz(A): 72004, nnz(LU): 682361, max. nnz: 3081301
PGMRES: pgmres(17): time: 1.25
estimated error = 3.3538e-03

*** checking for memory leaks ... no leaks.
```

Interpretation of the debugging information

The iterative solver class issues a line of debugging information for each call to the preconditioner (ilutp) and the solver (pgmres). These lines provide the following information:

PGMRES: ilutp:
- time: <consumed processor time in seconds>
- nnz(A): <structural nonzero elements of matrix A>

nnz(LU): <structural nonzero elements used by the factors L and U of A>
max. nnz: <maximum number of structural nonzero elements available with the current settings>

PGMRES: pgmres(<number of iterations>):
time: <consumed processor time in seconds>

Hints:

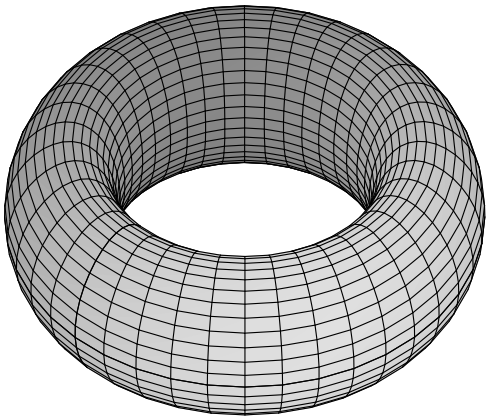
Always provide enough space for fill-in by setting LFil and Reserve to reasonable high values.

Play with DropTOL so that the sum of consumed processor time of ilutp and gmres becomes approximately a minimum. A good choice seem to be values for which
- (torfind) both times are almost equal
- (torcont) the time of ilutp is almost equal to the sum of the times of pgmres of each newton iteration (in torcont, for each corrector step the incomplete LU-factorisation is calculated only once, i.e. the different linear systems of each step are solved with the same preconditioner)

Set the tolerance of the linear solver to the maximal possible value, for which the newton process still converges nicely.
>

Create s plot of the computed torus. See the example "kawa" for the format of the data structure.

```
> data:=read_torus_data("lang", 1, 0):
  data:=select_torus_coords(data, 3,4,5):
  surfdata(data);
```



Create the parameter file expected by 'torcont' with settings
- problem name: "lang"
- run: 1

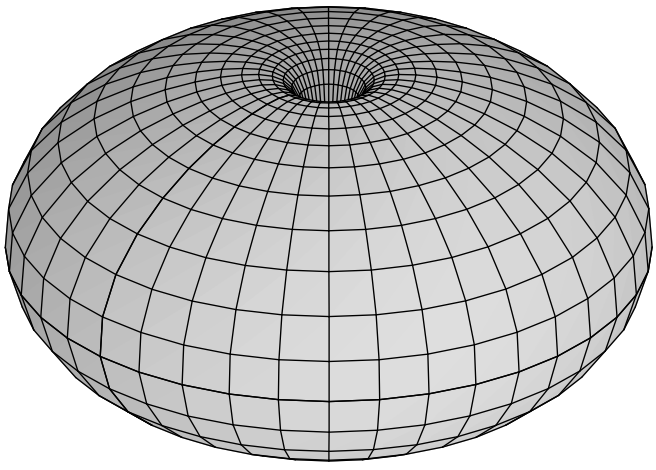
- continuation parameter: rho
- mesh: 40x40
- LFil and Reserve (see also ex. kawa): 200, 300
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.001
- max. value of rel residual of gmres (TOL): 1.0e-12
- don't print debugging information of iterative solver and ilu, LogFile: clog
- continuation interval: [0.15, 0.7]
- max. number of continuation steps: 200
- print every npr steps, npr: 5

and run torcont.

```
> create_tc_run ("lang", 1, continuer.param=rho,
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 200,
    linear_solver.Reserve = 300,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.001,
    linear_solver.TOL = 1.0e-12,
    linear_solver.LogFile = NULL, # clog,
    continuer.param_interval = [0.15, 0.7],
    continuer.ItMX = 200,
    npr = 5
  );
  torcont("lang", 1);
  STEP         PAR        ||x||          TOL    Period T1    Period T2 data
file
     0   5.500e-01   1.1830e+00   3.3538e-03   1.7952e+00   4.3115e+00 data/
lang.1/qpo0.dat
     5   5.627e-01   1.1764e+00   3.0264e-03   1.7952e+00   4.2625e+00 data/
lang.1/qpo1.dat
    10   5.796e-01   1.1676e+00   2.5121e-03   1.7952e+00   4.2007e+00 data/
lang.1/qpo2.dat
    15   5.934e-01   1.1602e+00   1.9799e-03   1.7952e+00   4.1527e+00 data/
lang.1/qpo3.dat
    20   6.040e-01   1.1546e+00   1.4345e-03   1.7952e+00   4.1173e+00 data/
lang.1/qpo4.dat
    25   6.111e-01   1.1507e+00   8.8021e-04   1.7952e+00   4.0939e+00 data/
lang.1/qpo5.dat
    30   6.149e-01   1.1486e+00   3.2054e-04   1.7952e+00   4.0820e+00 data/
lang.1/qpo6.dat
    31   6.149e-01   1.1486e+00   3.2054e-04   1.7952e+00   4.0820e+00 data/
lang.1/qpo7.dat
damped_newton: no convergence

  STEP         PAR        ||x||          TOL    Period T1    Period T2 data
file
     0   5.500e-01   1.1830e+00   3.3538e-03   1.7952e+00   4.3115e+00 data/
lang.1/qpo0.dat
     5   5.361e-01   1.1902e+00   3.6695e-03   1.7952e+00   4.3676e+00 data/
lang.1/qpo8.dat
    10   5.128e-01   1.2019e+00   4.1218e-03   1.7952e+00   4.4673e+00 data/
lang.1/qpo9.dat
    15   4.873e-01   1.2146e+00   4.5389e-03   1.7952e+00   4.5867e+00 data/
lang.1/qpo10.dat
    20   4.599e-01   1.2280e+00   4.9178e-03   1.7952e+00   4.7287e+00 data/
lang.1/qpo11.dat
    25   4.309e-01   1.2419e+00   5.2599e-03   1.7952e+00   4.8967e+00 data/
lang.1/qpo12.dat
    30   4.009e-01   1.2560e+00   5.5733e-03   1.7952e+00   5.0943e+00 data/
lang.1/qpo13.dat
    35   3.704e-01   1.2704e+00   5.8772e-03   1.7952e+00   5.3263e+00 data/
```

```
    35   3.704e-01   1.2704e+00   5.8772e-03   1.7952e+00   5.3263e+00 data/
lang.1/qpo14.dat
    40   3.399e-01   1.2847e+00   6.2052e-03   1.7952e+00   5.5978e+00 data/
lang.1/qpo15.dat
    45   3.101e-01   1.2989e+00   6.6068e-03   1.7952e+00   5.9147e+00 data/
lang.1/qpo16.dat
    50   2.814e-01   1.3128e+00   7.1439e-03   1.7952e+00   6.2837e+00 data/
lang.1/qpo17.dat
    55   2.545e-01   1.3265e+00   7.8840e-03   1.7952e+00   6.7123e+00 data/
lang.1/qpo18.dat
    60   2.297e-01   1.3398e+00   8.9780e-03   1.7952e+00   7.2090e+00 data/
lang.1/qpo19.dat
    65   2.074e-01   1.3528e+00   1.4906e-02   1.7952e+00   7.7831e+00 data/
lang.1/qpo20.dat
    70   1.877e-01   1.3654e+00   6.9333e-02   1.7952e+00   8.4457e+00 data/
lang.1/qpo21.dat
    75   1.706e-01   1.3776e+00   3.6438e-01   1.7952e+00   9.2095e+00 data/
lang.1/qpo22.dat
    80   1.560e-01   1.3895e+00   1.7298e+00   1.7952e+00   1.0090e+01 data/
lang.1/qpo23.dat
    83   1.485e-01   1.3964e+00   3.5604e+00   1.7952e+00   1.0683e+01 data/
lang.1/qpo24.dat

bifurcation diagramm written to file: 'data/lang.1/bd.dat'
output written to file:               'data/lang.1/torcont.log'

*** checking for memory leaks ... no leaks.

real  1:12:31.9
user  1:12:08.9
sys         1.1
```

```
> data:=read_torus_data("lang", 1, 19):
  data:=select_torus_coords(data, 3,4,5):
  surfdata(data);
```



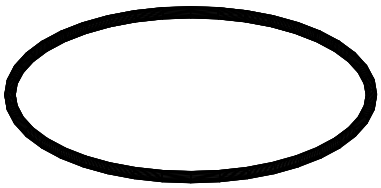
Here, we create an animation of the evolution of the torus under changes of the parameter rho.

At first, we read in all solutions given in a list (nums). The for-loop creates a set of plot-structures.

```
> nums:=[-($ -7..-0), $8..24]:
  an:=NULL:
  for i from 1 by 1 to nops(nums) do
    printf("reading graph %d\n", nums[i]);
    data:=read_torus_data("lang", 1, nums[i]):
    data:=select_torus_coords(data, 3,4,5):
    an:=an, surfdata({data}):
  od:
reading graph 7
reading graph 6
reading graph 5
reading graph 4
reading graph 3
reading graph 2
reading graph 1
reading graph 0
reading graph 8
reading graph 9
reading graph 10
reading graph 11
reading graph 12
reading graph 13
reading graph 14
reading graph 15
reading graph 16
reading graph 17
reading graph 18
reading graph 19
reading graph 20
reading graph 21
reading graph 22
reading graph 23
reading graph 24
```

Then, we display the tori together in a plot with the option insequence=true, which creates an animation. Select the plot and start the animation with the buttons in the toolbar.

```
> display([an], insequence=true);
```



[>

[>

Run 2: Continuation of Quasiperiodic Orbits: e=[0..0.1]

This run demonstrates the same as run 1, and in addition
 - how to obtain a start solution for a new run from a previous run at a specified parameter value.
Only this additional point is commented here.

Create a start solution for run 2 from a solution of run 1. Parameters:
 - problem name: "lang"
 - old run: 1
 - number of solution: 18
 - new run: 2
Because torcont does not print solutions at user-specified parameter values yet, this function may not be of much use at this time.

See the next execution group for an other way to obtain the required start solution.

```
> #copy_torus_solution("lang", 1, 18, 2);
test -d data/lang.2 || mkdir data/lang.2 ... OK
cp data/lang.1/qpo18.dat data/lang.2/qpo0.dat ... OK
```

This ‘alternative way´ of obtaining start solutions should become obsolete in the future.

We simply ‘misuse´ torfind to create the solution we require. We create a parameter file with the settings
 - problem name: "lang"

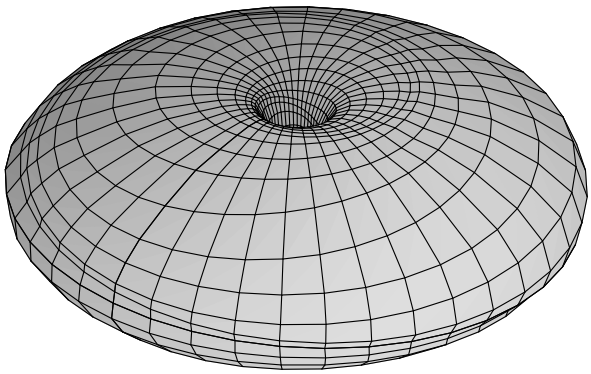
- (new!) run: 2
- start solution (format: data/<name>.<run>/qpo<num>.dat): "data/lang.1/qpo18.dat"
- set the value of rho to the required value: 0.25
- mesh: 40x40
- LFil, Reserve: 200, 250
- restart gmres after Restart iterations: 50
- max. number of iterations (gmres): 350
- dropping tolerance: 0.0025
- max. rel. residual (gmres): 1.0e-12
- print debugging information, LogFile: clog

and run torfind to compute the solution. torfind automatically writes the solution to the start solution of run 2.

```
> create_tf_run ("lang", 2, isol="data/lang.1/qpo18.dat",
    ode.rho = 0.25,
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 200,
    linear_solver.Reserve = 250,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.0025,
    linear_solver.TOL = 1.0e-12,
    linear_solver.LogFile = clog
  );
  torfind("lang", 2);
Iterat    D mpfung                  Normen                    Rechenzeit
  I SI       gamma       ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  5.3183e+01  1.5007e-01  0.0000e+00      0      0      0
PGMRES: ilutp: time: 6.7, nnz(A): 72004, nnz(LU): 1024023, max. nnz: 3081301
PGMRES: pgmres(20): time: 2.2
  1  1  1.0000e+00  5.3278e+01  8.1023e-03  4.3433e-01    0.3    3.1   28.5
PGMRES: ilutp: time: 6.4, nnz(A): 72004, nnz(LU): 998435, max. nnz: 3081301
PGMRES: pgmres(18): time: 1.9
  2  1  1.0000e+00  5.3277e+01  1.1367e-06  5.1322e-03    0.5    6.1   37.2
PGMRES: ilutp: time: 6.4, nnz(A): 72004, nnz(LU): 1003641, max. nnz: 3081301
PGMRES: pgmres(19): time: 2.0
  3  1  1.0000e+00  5.3277e+01  1.1644e-12  4.5306e-07    0.8    9.2   46.1

period T1 = 1.7951594779940942548e+00
period T2 = 6.7940976510276360756e+00
solution written to file 'data/lang.2/qpo0.dat'
estimating error ...
PGMRES: ilutp: time: 3.27, nnz(A): 72004, nnz(LU): 737751, max. nnz: 3081301
PGMRES: pgmres(18): time: 1.43
estimated error = 8.0399e-03

*** checking for memory leaks ... no leaks.
```

```
> create_tc_run ("lang", 2, continuer.param=epsilon,
    ode.rho = 0.25,
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 200,
    linear_solver.Reserve = 250,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.0025,
    linear_solver.TOL = 1.0e-12,
    continuer.param_interval = [0, 0.1],
```

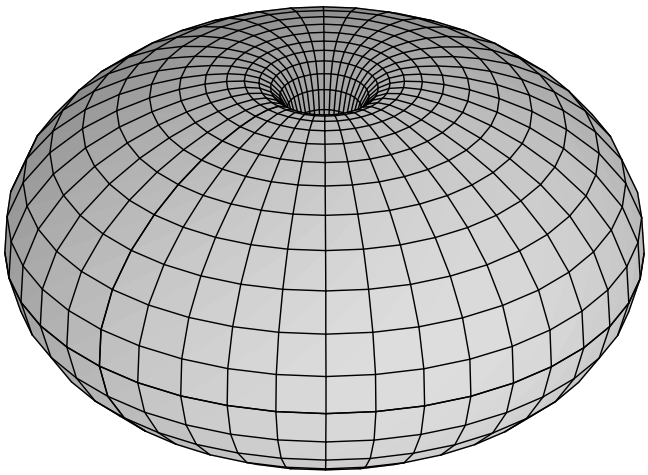
```
    continuer.ItMX = 20,
    npr = 1
  );
  torcont("lang", 2);
  STEP         PAR         ||x||           TOL    Period T1    Period T2 data
file
     0   0.000e+00   1.3288e+00   8.0399e-03   1.7952e+00   6.7941e+00 data/
lang.2/qpo0.dat
     1   9.332e-04   1.3288e+00   8.3369e-03   1.7952e+00   6.7941e+00 data/
lang.2/qpo1.dat
     2   2.706e-03   1.3288e+00   1.0275e-02   1.7952e+00   6.7942e+00 data/
lang.2/qpo2.dat
     3   6.075e-03   1.3289e+00   1.6476e-02   1.7952e+00   6.7946e+00 data/
lang.2/qpo3.dat
     4   1.074e-02   1.3289e+00   2.6758e-02   1.7952e+00   6.7956e+00 data/
lang.2/qpo4.dat
     5   1.541e-02   1.3290e+00   3.7694e-02   1.7952e+00   6.7971e+00 data/
lang.2/qpo5.dat
     6   2.010e-02   1.3290e+00   4.9045e-02   1.7952e+00   6.7990e+00 data/
lang.2/qpo6.dat
     7   2.483e-02   1.3291e+00   6.0820e-02   1.7952e+00   6.8012e+00 data/
lang.2/qpo7.dat
     8   2.963e-02   1.3292e+00   7.3100e-02   1.7952e+00   6.8035e+00 data/
lang.2/qpo8.dat
     9   3.455e-02   1.3293e+00   8.6002e-02   1.7952e+00   6.8058e+00 data/
lang.2/qpo9.dat
    10   3.966e-02   1.3294e+00   9.9691e-02   1.7952e+00   6.8080e+00 data/
lang.2/qpo10.dat
    11   4.507e-02   1.3296e+00   1.1442e-01   1.7952e+00   6.8100e+00 data/
lang.2/qpo11.dat
    12   5.091e-02   1.3297e+00   1.3066e-01   1.7952e+00   6.8115e+00 data/
lang.2/qpo12.dat
    13   5.739e-02   1.3298e+00   1.4933e-01   1.7952e+00   6.8123e+00 data/
lang.2/qpo13.dat
    14   6.479e-02   1.3298e+00   1.7229e-01   1.7952e+00   6.8122e+00 data/
lang.2/qpo14.dat
    15   7.355e-02   1.3299e+00   2.0330e-01   1.7952e+00   6.8107e+00 data/
lang.2/qpo15.dat
    16   8.429e-02   1.3299e+00   2.4993e-01   1.7952e+00   6.8071e+00 data/
lang.2/qpo16.dat
    17   9.570e-02   1.3298e+00   3.1286e-01   1.7952e+00   6.8012e+00 data/
lang.2/qpo17.dat
    18   1.051e-01   1.3297e+00   3.7633e-01   1.7952e+00   6.7951e+00 data/
lang.2/qpo18.dat

  STEP         PAR         ||x||           TOL    Period T1    Period T2 data
file
     0   0.000e+00   1.3288e+00   8.0399e-03   1.7952e+00   6.7941e+00 data/
lang.2/qpo0.dat
     1  -9.332e-04   1.3288e+00   8.3369e-03   1.7952e+00   6.7941e+00 data/
lang.2/qpo19.dat

bifurcation diagramm written to file: 'data/lang.2/bd.dat'
output written to file:               'data/lang.2/torcont.log'

*** checking for memory leaks ... no leaks.

real    13:13.2
user    13:10.0
sys         0.8
> data:=read_torus_data("lang", 2, 10):
  data:=select_torus_coords(data, 3,4,5):
  surfdata(data);
```



```
> nums:=[$0..18]:
  an:=NULL:
  for i from 1 by 1 to nops(nums) do
    printf("reading graph %d\n", nums[i]);
    data:=read_torus_data("lang", 2, nums[i]):
    data:=select_torus_coords(data, 3,4,5):
    an:=an, surfdata({data}):
  od:
```

$$nums := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]$$

```
reading graph 0
reading graph 1
reading graph 2
reading graph 3
reading graph 4
reading graph 5
reading graph 6
reading graph 7
reading graph 8
reading graph 9
reading graph 10
reading graph 11
reading graph 12
reading graph 13
reading graph 14
reading graph 15
reading graph 16
reading graph 17
reading graph 18
> display([an], insequence=true);
```



Two coupled Van der Pol Oscillators

This famous system must appear as an example of a package for computation and continuation of invariant tori. Here it comes:

$$\left(\frac{\partial^2}{\partial t^2}x\right)+\varepsilon\,(x^2-1)\left(\frac{\partial}{\partial t}x\right)+x=\beta\,(y-x)$$

$$\left(\frac{\partial^2}{\partial t^2}y\right)+\varepsilon\,(y^2-1)\left(\frac{\partial}{\partial t}y\right)+y=\beta\,(x-y)-\delta\,y$$

The system is of interest for (different) fixed values of epsilon and for beta, delta > 0. Here, we do only two simple continuations of tori with respect to either beta (run 1) or delta (run 2).

Attention! Before starting the computations, please set the paths in the ‘read´ and the ‘currentdir´ commands correctly. Ignore the error message issued by ‘mkdir´.

```
> restart;
  with(plots):
  with(process):
  read "/export/fschild/maple/contpack.m":
  currentdir("/export/fschild/examples/vdp");
  mkdir("data");
  currentdir();
Warning, the name changecoords has been redefined

Error, (in mkdir) directory exists and is not empty
```

"/export/fschild/examples/vdp"

```
>
>
>
>
```

Definition of the System and Creation of the Shared Object

The constants and parameters are defined as a list of ‘name = value´ pairs.
The system is defined as a function taking a list and a number as arguments and returning a list of expressions.

```
> Constants := []:
  Params    := [e=0.3, d=0.2, b=0.001]:

  VDP := (x,t) -> [
    x[2],
    -x[1]+b*(x[3]-x[1])+e*(1-x[1]^2)*x[2],
    x[4],
    -(1+d)*x[3]+b*(x[1]-x[3])+e*(1-x[3]^2)*x[4]
  ]:
```

Create a shared object by calling codegen and then compiling and linking.
The compiler options used are for Solaris. You may need different options.

```
> create_ode("vdp", VDP, Constants, Params,
    compiler = "gcc -fPIC",
```

```
    linker   = "gcc -fPIC -shared"
  );
codegen < vdp.ode > vdp.c ... OK
gcc -fPIC -c -o vdp.o vdp.c ... OK
gcc -fPIC -shared -o vdp.so vdp.o ... OK
```

Definition of the Start Solutions

Define initial approximations to the quasiperiodic solution.
This function is a guess.

This torus function(s) must always be 2*Pi-periodic in each argument.

```
> isol := (t,th) -> [
    2*sin(t), 2*cos(t),
    2*sin(th), 2.19*cos(th)
  ]:
```

Write the initial solution to disk on a 40x40 mesh.

Parameters:
- problem name: "vdp"
- run: 1
- name of the function calculating initial solution values: isol
- T1: 6.3
- T2: 5.8
- mesh: 40x40

```
> write_tss("vdp", 1, isol, 6.3, 5.8, 40, 40);
                                  "vdp_qpo1.dat"
```

Run 1: Continuation of Quasiperiodic Orbits: b=[0..0.22]

This run demonstrates, how to
- run torfind and torcont.
- tune the step size control of the continuer
- print and interpret debugging information of the continuer.
- create 3D-plots and animations.

Create the parameter file expected by 'torfind' with settings
- problem name: "vdp"
- run: 1
- initial solution file: "vdp_qpo1.dat"
- mesh: 40x40
- LFil and Reserve (see also ex. kawa): 250, 300
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.01
- max. value of rel residual of gmres (TOL): 1.0e-7
- don't print debugging information of iterative solver and ilu, LogFile: NULL

and run torfind.

```
> create_tf_run ("vdp", 1, isol="vdp_qpo1.dat",
```

```
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 250,
    linear_solver.Reserve = 300,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.01,
    linear_solver.TOL = 1.0e-7,
    linear_solver.LogFile = NULL # clog
  );
  torfind("vdp", 1);
```

```
Iterat    D mpfung                  Normen                    Rechenzeit
  I SI       gamma        ||x||       ||f|| gamma*||d||   F(x)  DF(x)  solve

  0  0  0.0000e+00  1.1593e+02  2.5166e+01  0.0000e+00      0      0      0
  1  1  1.0000e+00  1.1626e+02  3.1900e-01  9.4264e+00    0.3    4.4   64.1
  2  1  1.0000e+00  1.1612e+02  2.1970e-04  1.6002e-01    0.6    8.6   94.4
  3  1  1.0000e+00  1.1612e+02  4.7285e-08  4.8280e-04    1.0   12.8  126.0
  4  1  1.0000e+00  1.1612e+02  2.2302e-10  5.3201e-05    1.3   17.0  158.2

period T1 = 6.3150750482154807131e+00
period T2 = 5.7599935747348069981e+00
solution written to file 'data/vdp.1/qpo0.dat'

*** checking for memory leaks ... no leaks.
```

Create the parameter file expected by 'torcont' with settings
- problem name: "vdp"
- run: 1
- continuation parameter (beta): b
- mesh: 40x40
- LFil and Reserve (see also ex. kawa): 250, 300
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.01
- max. value of rel residual of gmres (TOL): 1.0e-7
- don't print debugging information of iterative solver and ilu, LogFile: NULL
- continuation interval: [0, 0.22]
- max. angle between the tangencies of two successive continuation steps (degree): 20
- initial continuation step size: 10
- max. continuation step size: 25
- min. continuation step size: 1
- max. number of continuation steps: 25
- print debugging information (continuer), LogFile: clog
- print every npr steps, npr: 3

and run torcont.

```
> create_tc_run ("vdp", 1, continuer.param=b,
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 250,
    linear_solver.Reserve = 300,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.01,
    linear_solver.TOL = 1.0e-7,
    linear_solver.LogFile = NULL, # clog,
    continuer.param_interval = [0, 0.22],
```

```
    continuer.Alpha = 20,
    continuer.h0 = 10,
    continuer.h_max = 25,
    continuer.h_min = 1,
    continuer.ItMX = 25,
    continuer.LogFile = clog,
    npr = 3
  );
  torcont("vdp", 1);
continuer::initialize:
h=1.0000e+01, h_max=2.5000e+01, h_min=1.0000e+00
h_fac_min=5.0000e-01, h_fac_max=2.0000e+00, MaxDiff=2.5000e-01, alpha=2.0000
e+01, gamma=9.5000e-01
  STEP         PAR        ||x||          TOL    Period T1    Period T2 data
file
     0   1.000e-03   2.9027e+00   1.1089e-02   6.3151e+00   5.7600e+00 data/
vdp.1/qpo0.dat
continuer::step:
norm(x)=1.1659e+02, norm(v-x)=7.6098e-01, rel_abs_diff=6.4717e-03
beta=8.7267e+00, h_fac1=3.8630e+01, h_fac2=2.2824e+00
h=1.9000e+01
continuer::step:
norm(x)=1.1664e+02, norm(v-x)=3.0422e+00, rel_abs_diff=2.5860e-02
beta=1.8852e+01, h_fac1=9.6676e+00, h_fac2=1.0603e+00
h=1.9138e+01
continuer::step:
norm(x)=1.1488e+02, norm(v-x)=4.5831e+00, rel_abs_diff=3.9551e-02
beta=3.0713e+01, h_fac1=6.3209e+00, h_fac2=6.5572e-01
h=1.1922e+01
     3   1.183e-01   2.8717e+00   5.1927e-01   6.0004e+00   5.4889e+00 data/
vdp.1/qpo1.dat
continuer::step:
norm(x)=1.1140e+02, norm(v-x)=5.0576e+00, rel_abs_diff=4.4997e-02
beta=5.3340e+01, h_fac1=5.5559e+00, h_fac2=3.8688e-01
h=5.6629e+00
continuer::step:
norm(x)=1.0951e+02, norm(v-x)=2.5570e+00, rel_abs_diff=2.3138e-02
beta=3.4501e+01, h_fac1=1.0805e+01, h_fac2=5.8556e-01
h=3.1502e+00
continuer::step:
norm(x)=1.0882e+02, norm(v-x)=8.0444e-01, rel_abs_diff=7.3251e-03
beta=2.8645e+01, h_fac1=3.4129e+01, h_fac2=7.0195e-01
h=2.1007e+00
     6   1.716e-01   2.7202e+00   2.2653e+00   5.9185e+00   5.3240e+00 data/
vdp.1/qpo2.dat
continuer::step:
norm(x)=1.0855e+02, norm(v-x)=3.8154e-01, rel_abs_diff=3.4827e-03
beta=1.8878e+01, h_fac1=7.1783e+01, h_fac2=1.0589e+00
h=2.1131e+00
continuer::step:
norm(x)=1.0842e+02, norm(v-x)=2.9877e-01, rel_abs_diff=2.7304e-03
beta=1.3773e+01, h_fac1=9.1562e+01, h_fac2=1.4482e+00
h=2.9072e+00
continuer::step:
norm(x)=1.0840e+02, norm(v-x)=2.5602e-01, rel_abs_diff=2.3402e-03
beta=9.0787e+00, h_fac1=1.0683e+02, h_fac2=2.1941e+00
h=5.5237e+00
     9   1.784e-01   2.7097e+00   3.4224e+00   5.9041e+00   5.2836e+00 data/
vdp.1/qpo3.dat
continuer::step:
norm(x)=1.0861e+02, norm(v-x)=5.0460e-01, rel_abs_diff=4.6037e-03
beta=9.6509e+00, h_fac1=5.4304e+01, h_fac2=2.0643e+00
h=1.0495e+01
continuer::step:
norm(x)=1.0935e+02, norm(v-x)=1.3745e+00, rel_abs_diff=1.2455e-02
beta=1.4659e+01, h_fac1=2.0072e+01, h_fac2=1.3611e+00
h=1.3571e+01
continuer::step:
norm(x)=1.1050e+02, norm(v-x)=2.2317e+00, rel_abs_diff=2.0016e-02
beta=1.8470e+01, h_fac1=1.2490e+01, h_fac2=1.0820e+00
h=1.3949e+01
    12   1.878e-01   2.7621e+00   1.2184e+01   5.8412e+00   5.2039e+00 data/
vdp.1/qpo4.dat
```

```
continuer::step:
norm(x)=1.1159e+02, norm(v-x)=2.3874e+00, rel_abs_diff=2.1205e-02
beta=1.9306e+01, h_fac1=1.1790e+01, h_fac2=1.0356e+00
h=1.3723e+01
continuer::step:
norm(x)=1.1243e+02, norm(v-x)=2.3620e+00, rel_abs_diff=2.0823e-02
beta=1.9235e+01, h_fac1=1.2006e+01, h_fac2=1.0394e+00
h=1.3551e+01
continuer::step:
norm(x)=1.1301e+02, norm(v-x)=2.2373e+00, rel_abs_diff=1.9625e-02
beta=1.8554e+01, h_fac1=1.2739e+01, h_fac2=1.0772e+00
h=1.3867e+01
    15   2.013e-01   2.8249e+00   2.8938e-01   5.8351e+00   5.1614e+00 data/
vdp.1/qpo5.dat
continuer::step:
norm(x)=1.1334e+02, norm(v-x)=2.3340e+00, rel_abs_diff=2.0412e-02
beta=1.9131e+01, h_fac1=1.2247e+01, h_fac2=1.0450e+00
h=1.3766e+01
continuer::step:
norm(x)=1.1341e+02, norm(v-x)=2.3952e+00, rel_abs_diff=2.0935e-02
beta=1.9711e+01, h_fac1=1.1942e+01, h_fac2=1.0145e+00
h=1.3267e+01
continuer::step:
norm(x)=1.1322e+02, norm(v-x)=2.2104e+00, rel_abs_diff=1.9352e-02
beta=1.8712e+01, h_fac1=1.2918e+01, h_fac2=1.0681e+00
h=1.3463e+01
    18   2.035e-01   2.8302e+00   3.1390e-01   5.8404e+00   5.1589e+00 data/
vdp.1/qpo6.dat
continuer::step:
norm(x)=1.1278e+02, norm(v-x)=2.1901e+00, rel_abs_diff=1.9248e-02
beta=1.8354e+01, h_fac1=1.2988e+01, h_fac2=1.0888e+00
h=1.3926e+01
continuer::step:
norm(x)=1.1207e+02, norm(v-x)=2.3721e+00, rel_abs_diff=2.0979e-02
beta=1.9381e+01, h_fac1=1.1917e+01, h_fac2=1.0316e+00
h=1.3648e+01
continuer::step:
norm(x)=1.1112e+02, norm(v-x)=2.3814e+00, rel_abs_diff=2.1240e-02
beta=1.9265e+01, h_fac1=1.1770e+01, h_fac2=1.0378e+00
h=1.3455e+01
    21   1.901e-01   2.7776e+00   2.2214e+00   5.8322e+00   5.1908e+00 data/
vdp.1/qpo7.dat
continuer::step:
norm(x)=1.1000e+02, norm(v-x)=2.2160e+00, rel_abs_diff=1.9963e-02
beta=1.8481e+01, h_fac1=1.2523e+01, h_fac2=1.0814e+00
h=1.3823e+01
continuer::step:
norm(x)=1.0889e+02, norm(v-x)=2.3212e+00, rel_abs_diff=2.1124e-02
beta=1.8871e+01, h_fac1=1.1835e+01, h_fac2=1.0593e+00
h=1.3910e+01
continuer::step:
norm(x)=1.0849e+02, norm(v-x)=2.7439e+00, rel_abs_diff=2.5059e-02
beta=2.8563e+01, h_fac1=9.9763e+00, h_fac2=7.0393e-01
h=9.3019e+00
    24   1.751e-01   2.7121e+00   7.5788e-01   5.9136e+00   5.3052e+00 data/
vdp.1/qpo8.dat
continuer::step:
norm(x)=1.1129e+02, norm(v-x)=6.5002e+00, rel_abs_diff=5.7886e-02
beta=6.0485e+01, h_fac1=4.3188e+00, h_fac2=3.4477e-01
h=4.4184e+00
    25   1.539e-01   2.7820e+00   1.5968e+00   5.9347e+00   5.3946e+00 data/
vdp.1/qpo9.dat

continuer::initialize:
h=-1.0000e+01, h_max=2.5000e+01, h_min=1.0000e+00
h_fac_min=5.0000e-01, h_fac_max=2.0000e+00, MaxDiff=2.5000e-01, alpha=2.0000
e+01, gamma=9.5000e-01
  STEP         PAR        ||x||          TOL    Period T1    Period T2 data
file
     0   1.000e-03   2.9027e+00   1.1089e-02   6.3151e+00   5.7600e+00 data/
vdp.1/qpo0.dat
continuer::step:
norm(x)=1.1536e+02, norm(v-x)=7.7010e-01, rel_abs_diff=6.6182e-03
beta=8.8840e+00, h_fac1=3.7774e+01, h_fac2=2.2421e+00
h=-1.9000e+01
     1  -2.167e-02   2.8838e+00   3.4924e-02   6.3899e+00   5.8156e+00 data/
vdp.1/qpo10.dat
```

```
bifurcation diagramm written to file: 'data/vdp.1/bd.dat'
output written to file:               'data/vdp.1/torcont.log'

*** checking for memory leaks ... no leaks.

real  1:33:32.5
user  1:33:29.8
sys         1.0
```

Interpretation of the debugging information

The continuer class debugging information for two function calls, continuer::initialize and continuer::step. The output has the following meaning:

continuer::initialize:

- h : the initial continuation step size (the sign decides the direction of the continuation)
- h_max : the max. continuation step size
- h_min : the min. continuation step size
- h_fac_min : the min. factor for step size adaption
- f_fac_max : the max. factor for step size adaption
- MaxDiff : the max. value of ||v-x||/(1+||x||) where v is the predicted and x the corrected solution
- alpha : the maximum angle between to tangencies of two successice solutions
- gamma : a security factor

The step size is adapted by h=gamma*h_fac*h with h_fac_min <= h_fac <= h_fac_max. h_fac is computed, so that two solutions have a (rel-abs-) distance of at most MaxDiff and the angle between two successive tangencies is smaller than alpha. We do not take the iteration number of the corrector into account. Thereby, it is possible to use black box-correctors for the continuation, for instance from the package minpack, which do not provide iteration numbers (or where they do not make too much sense). A clever choice of these parameters can speed up your continuation dramatically. For hard problems (like this example) you should adjust these parameters. For checking the success of your adaption, use the output of

continuer::step:

- norm(x) : norm of the solution vector, this is the non-normalized euclidian norm
- norm(v-x) : norm of the difference between predicted and corrected solution
- rel_abs_diff : norm(v-x)/(1+norm(x)), this value is used to estimate h_fac1 in comparison with MaxDiff
- beta : the actual angle between two successive tangencies, this value is used to estimate h_fac2 in comparison with alpha
- h_fac1 : the factor for changing the step size; estimated, so that rel_abs_err may become equal to MaxDiff
- h_fac2 : the factor for changing the step size; estimated, so that beta may become equal to alpha
- h : the new step size which will be used in the next step

Hints:

MaxDiff probably needs not to be changed. 0.25 is mostly never met.

The angle is a quite good measure of the nonlinearity of your problem and usually dominates the step size control. Change the value of Alpha with care! If you enlarge Alpha, always set h_max to a value, wich is not too large.

Usually, the debigging information helps you very much to find out, why the continuer chooses a small step size. In this case, you may need to adapt the default values.

>

Here, we create an animation of the evolution of the torus under changes of the parameter b (beta).
At first, we read in all solutions given in a list (nums). The for-loop creates two sequences of plot-structures.

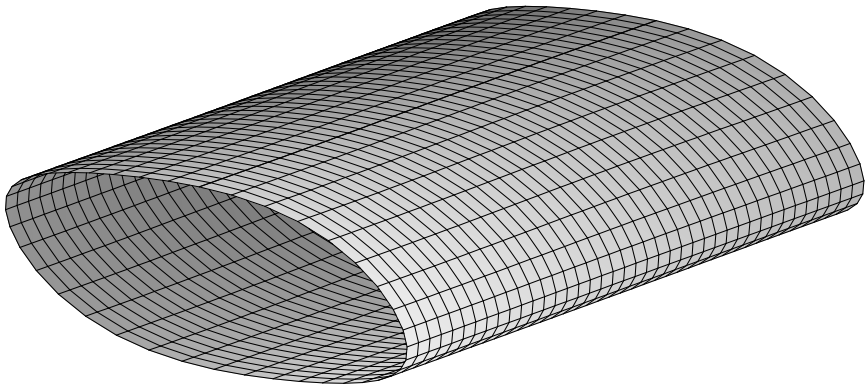
For plotting, we choose (θ_2, x, $\frac{\partial}{\partial t}x$) for an1 and ($\theta_1$, y, $\frac{\partial}{\partial t}y$) for an2.

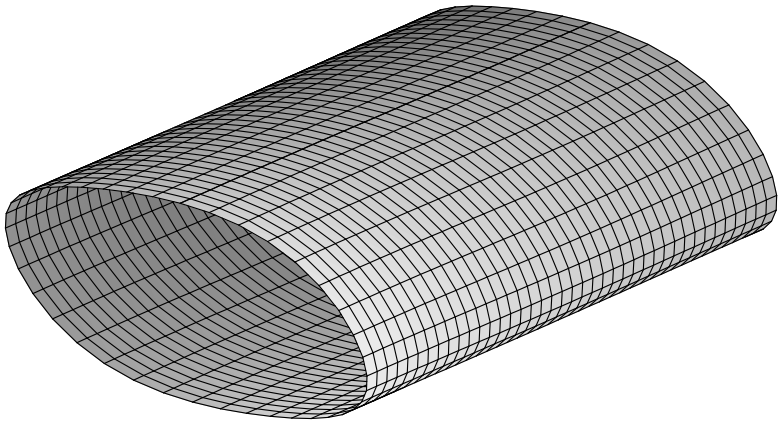
```
> nums:=[$0..9]:
  an1:=NULL : an2:=NULL :
  for i from 1 by 1 to nops(nums) do
    printf("reading graph %d\n", nums[i]);
    data:=read_torus_data("vdp", 1, nums[i]):
    data1:=select_torus_coords(data, 2,3,4):
    data2:=select_torus_coords(data, 1,5,6):
    an1:=an1, surfdata({data1}):
    an2:=an2, surfdata({data2}):
  od:
reading graph 0
reading graph 1
reading graph 2
reading graph 3
reading graph 4
reading graph 5
reading graph 6
reading graph 7
reading graph 8
reading graph 9
```

We create two animations, which show the evolution of the torus as the continuation progresses. For beta near 0.2035, the torus disappears in a Hopf bifurcation to a periodic orbit. The angle-coordinate $\theta_1$becomes obsolete at this point (as the dimension of the manifold drops from 2 (torus) to 1 (periodic orbit)). Therefore, the plot over θ_1 seems to be still a torus, but it is rather the same periodic orbit plottet for each value of θ_1.

There is an interesting additional observation. The Hopf-point seems also to be a limit-point of the torus-curve! This impression is aided by the second animation, which clearly shows different pictures for each of the branches. (In addition to the continuation process, which turns around near this Hopf-point. Further, for beta=0, the torus is not unique. Thats the reason for choosing beta=0.001 for the initial solution. This all together indicates, that there is a closed loop of tori with limit points for beta=0 and at the Hopf-points, which depend on delta.)

```
> display([an1], insequence=true);
  display([an2], insequence=true);
```





[>

[>

Run 2: Continuation of Quasiperiodic Orbits: b=0.1, d=[0..10]

This run demonstrates the same as run 1 without debugging information, and in addition
- how to obtain a start solution from a previous run at a specified parameter value.

Only this additional point is commented here at length.

Create a start solution for run 2 from a solution of run 1. Parameters:
- problem name: "vdp"
- old run: 1
- number of solution: 1
- new run: 2

Because torcont does not print solutions at user-specified parameter values yet, this function may not be of much use at this time.

See the next execution group on how to obtain the required start solution.

```
> #copy_torus_solution("vdp", 1, 1, 2);
test -d data/vdp.2 || mkdir data/vdp.2 ... OK
cp data/vdp.1/qpo1.dat data/vdp.2/qpo0.dat ... OK
```

This ‘alternative way´ of obtaining start solutions should become obsolete in the future.

We simply ‘misuse´ torfind to create the solution we require. We create a parameter file with the settings
- problem name: "vdp"
- run: 2
- initial solution (format: data/<name>.<run>/qpo<num>.dat): data/vdp.1/qpo1.dat
- mesh: 40x40
- LFil and Reserve (see also ex. kawa): 250, 300
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 350
- dropping tolerance for ilu: 0.01
- max. value of rel residual of gmres (TOL): 1.0e-7
- don’t print debugging information of iterative solver and ilu, LogFile: NULL

and run torfind to compute the solution. torfind automatically writes the solution to the start solution of run 2.

```
> create_tf_run ("vdp", 2, isol="data/vdp.1/qpo1.dat",
    ode.b = 0.1,
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 250,
    linear_solver.Reserve = 300,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 350,
    linear_solver.DropTOL = 0.01,
    linear_solver.TOL = 1.0e-7,
    linear_solver.LogFile = NULL # clog
  );
  torfind("vdp", 2);
Iterat    D mpfung                  Normen                    Rechenzeit
  I SI       gamma       ||x||       ||f|| gamma*||d||   F(x)   DF(x)   solve

  0  0  0.0000e+00  1.1488e+02  1.9697e+00  0.0000e+00      0      0       0
  1  1  1.0000e+00  1.1634e+02  6.0309e-01  8.2195e+00    0.3    4.3   134.6
  2  1  1.0000e+00  1.1595e+02  1.1565e-01  3.4337e+00    0.6    8.5   228.6
  3  1  1.0000e+00  1.1583e+02  2.6079e-03  4.9449e-01    1.0   12.7   322.4
  4  1  1.0000e+00  1.1582e+02  7.9621e-06  2.7182e-02    1.3   16.9   418.3
  5  1  1.0000e+00  1.1582e+02  1.3183e-08  5.4643e-05    1.6   21.0   511.1

period T1 = 6.042190594389074576e+00
period T2 = 5.530865081657868387e+00
solution written to file 'data/vdp.2/qpo0.dat'
```

```
estimating error ...
estimated error = 1.8118e-01

*** checking for memory leaks ... no leaks.
```

Create the parameter file expected by 'torcont' with settings
- problem name: "vdp"
- run: 2
- continuation parameter (beta): b
- set the value of b (beta) to the desired value: 0.1
- mesh: 40x40
- LFil and Reserve (see also ex. kawa): 500, 500
- use a Krylov subspace of at most Restart dimensions, Restart: 50
- max. number of iterations of gmres (ItMX): 500
- dropping tolerance for ilu: 0.005
- max. value of rel residual of gmres (TOL): 1.0e-9
- continuation interval: [0, 10]
- max. number of continuation steps: 80
- max. angle between the tangencies of two successive continuation steps (degree): 10
- initial continuation step size: 1
- max. continuation step size: 5
- min. continuation step size: 0.1
- don't print debugging information (continuer): outcommented
- print every npr steps, npr: 5

and run torcont.

```
> create_tc_run ("vdp", 2, continuer.param=d,
    ode.b = 0.1,
    discretisation_points1 = 40,
    discretisation_points2 = 40,
    linear_solver.LFil = 500,
    linear_solver.Reserve = 500,
    linear_solver.Restart = 50,
    linear_solver.ItMX = 500,
    linear_solver.DropTOL = 0.005,
    linear_solver.TOL = 1.0e-9,
    continuer.param_interval = [0, 10],
    continuer.ItMX = 80,
    continuer.Alpha = 10,
    continuer.h0 = 1,
    continuer.h_max = 5,
    continuer.h_min = 0.1,
  #  continuer.LogFile = clog,
    npr = 5
  );
  torcont("vdp", 2);
  STEP         PAR        ||x||          TOL    Period T1    Period T2 data
file
     0   2.000e-01   2.8953e+00   1.8118e-01   6.0422e+00   5.5309e+00 data/
vdp.2/qpo0.dat
     5   3.362e-01   2.9568e+00   5.9696e-02   6.0470e+00   5.2557e+00 data/
vdp.2/qpo1.dat
    10   1.020e+00   3.2137e+00   1.2898e-02   6.0418e+00   4.3208e+00 data/
vdp.2/qpo2.dat
    15   2.405e+00   3.6314e+00   1.0497e-02   6.0321e+00   3.3598e+00 data/
vdp.2/qpo3.dat
    20   4.229e+00   4.1058e+00   1.0368e-02   6.0277e+00   2.7241e+00 data/
vdp.2/qpo4.dat
    25   6.447e+00   4.6153e+00   1.1818e-02   6.0256e+00   2.2886e+00 data/
vdp.2/qpo5.dat
```

```
    30   8.518e+00   5.0441e+00   2.1908e-01   6.0256e+00   2.0269e+00 data/
vdp.2/qpo6.dat
    35   8.586e+00   5.0556e+00   4.3107e-01   6.0255e+00   2.0198e+00 data/
vdp.2/qpo7.dat
    40   8.611e+00   5.0582e+00   5.9263e-01   6.0234e+00   2.0172e+00 data/
vdp.2/qpo8.dat
    45   8.619e+00   5.0591e+00   6.4788e-01   6.0225e+00   2.0164e+00 data/
vdp.2/qpo9.dat
    50   8.617e+00   5.0589e+00   7.2793e-01   6.0227e+00   2.0166e+00 data/
vdp.2/qpo10.dat
    55   8.588e+00   5.0559e+00   2.8561e+00   6.0253e+00   2.0195e+00 data/
vdp.2/qpo11.dat
    60   8.515e+00   5.0436e+00   2.0275e-01   6.0255e+00   2.0272e+00 data/
vdp.2/qpo12.dat
    65   8.314e+00   5.0038e+00   1.0920e-01   6.0251e+00   2.0488e+00 data/
vdp.2/qpo13.dat
sparskit::PGMRES::pgmres: no convergence

  STEP         PAR        ||x||          TOL    Period T1    Period T2 data
file
     0   2.000e-01   2.8953e+00   1.8118e-01   6.0422e+00   5.5309e+00 data/
vdp.2/qpo0.dat
     5   1.430e-01   2.8685e+00   1.6337e+01   6.0366e+00   5.6612e+00 data/
vdp.2/qpo14.dat
    10   1.005e-01   2.8465e+00   3.8479e+01   6.0293e+00   5.7655e+00 data/
vdp.2/qpo15.dat
    15   9.374e-02   2.8343e+00   2.3998e+02   6.0304e+00   5.7819e+00 data/
vdp.2/qpo16.dat
    20   9.252e-02   2.8267e+00   2.2195e+01   6.0311e+00   5.7837e+00 data/
vdp.2/qpo17.dat
    25   9.326e-02   2.8238e+00   7.4058e+00   6.0297e+00   5.7801e+00 data/
vdp.2/qpo18.dat
    30   9.676e-02   2.8302e+00   5.9467e+00   6.0252e+00   5.7679e+00 data/
vdp.2/qpo19.dat
    35   9.752e-02   2.8345e+00   1.8941e+00   6.0222e+00   5.7574e+00 data/
vdp.2/qpo20.dat
    40   9.471e-02   2.8383e+00   2.6917e+00   6.0186e+00   5.7555e+00 data/
vdp.2/qpo21.dat
    45   8.979e-02   2.8362e+00   5.1114e+00   6.0156e+00   5.7589e+00 data/
vdp.2/qpo22.dat
    50   8.366e-02   2.8195e+00   3.6437e+00   6.0182e+00   5.7703e+00 data/
vdp.2/qpo23.dat
    55   8.042e-02   2.8008e+00   1.2183e+01   6.0249e+00   5.7797e+00 data/
vdp.2/qpo24.dat
time: command terminated abnormally.

real  4:28:20.5
user  4:27:33.4
sys         2.4
```

Torcont was killed here, hence the message of time.

```
>
```

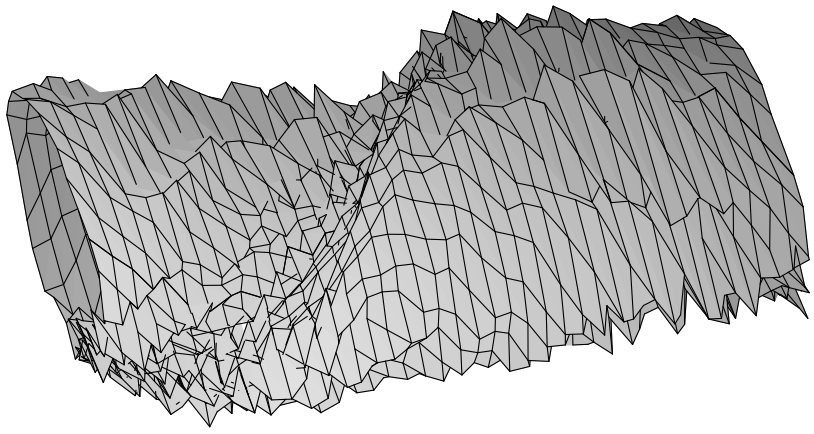
Here, we create two animations of the torus. At first, we read in all the solutions given in a list (nums). The for-loop creates two sequences of plot-structures (an1 and an2). See example "kawa" for the format of the torus data structure.

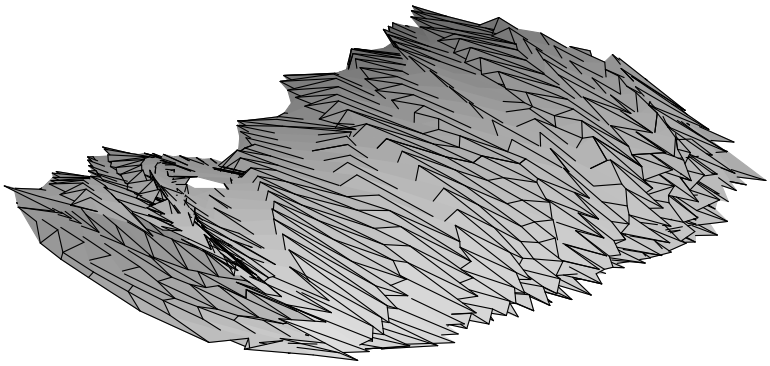
```
> nums:=[-($ -24 .. -14), $ 0 .. 13]:
  an1:=NULL : an2:=NULL :
  for i from 1 by 1 to nops(nums) do
    printf("reading graph %d\n", nums[i]);
    data:=read_torus_data("vdp", 2, nums[i]):
    data1:=select_torus_coords(data, 2,3,4):
    data2:=select_torus_coords(data, 1,5,6):
    an1:=an1, surfdata({data1}):
    an2:=an2, surfdata({data2}):
  od:
reading graph 24
reading graph 23
reading graph 22
reading graph 21
reading graph 20
```

```
reading graph 19
reading graph 18
reading graph 17
reading graph 16
reading graph 15
reading graph 14
reading graph 0
reading graph 1
reading graph 2
reading graph 3
reading graph 4
reading graph 5
reading graph 6
reading graph 7
reading graph 8
reading graph 9
reading graph 10
reading graph 11
reading graph 12
reading graph 13
```

In this animation you can see some tori, which appear not smooth. This is due to (strong) resonances, which occour at different subintervals of our continuation interval. See the section ‘Bifurcation Diagram´ for further information.

```
> display([an1], insequence=true);
  display([an2], insequence=true);
```





[>

Bifurcation Diagram

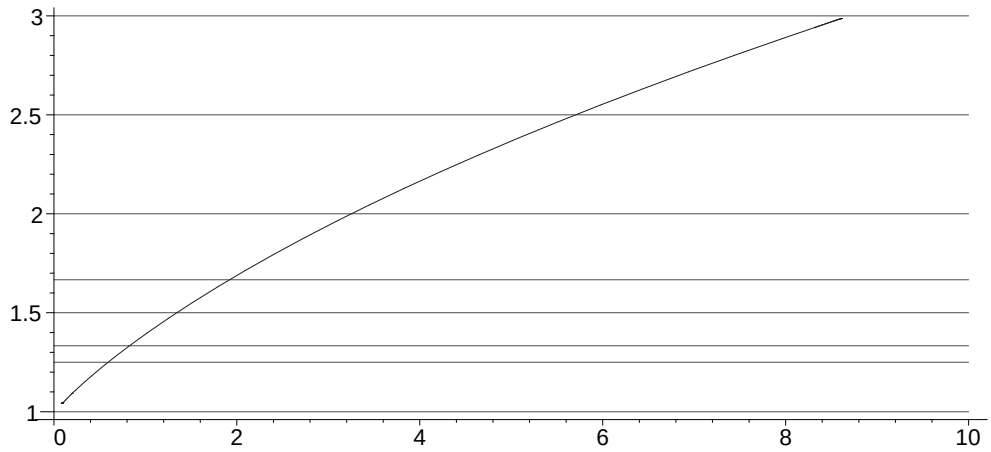
Within the continuation interval, there occour some strong resonances. For easy detection, we plot the rotation number together with lines of critical values over the parameter d (delta). At points, where the rotation number crosses such a critical line, a strong resonance occurs. The rotation number then remains constant over some interval. This effect is known as a phase-lock. In our graph, we can observe this phase-lock for the 1:1 and the 1:3 resonance, where our method breaks down (which here is mostly due to the crude discretisation). The other resonances are numerically not observable, because for epsilon=0.3 and beta=0.1, these phase-locking-intervals are very narrow. For a finer mesh and higher values of epsilon and beta, these are observed also.

We read the bifurcation diagram into data1. data1 then contains two lists of data, one for the forward and one for the backward continuation. From both lists we extract the parameter and the two basic periods. By dividing both periods by each other, we obtain the rotation number. These is plotted (blue) together with the lines of strong resonances (red), which are possible for rotation numbers in the interval [1,3], over the parameter d (delta). Weak(er) resonances are not considered here (and also not ‘really´ observed numerically).

See example "kawa" for the format of the bifurcation diagram structure.

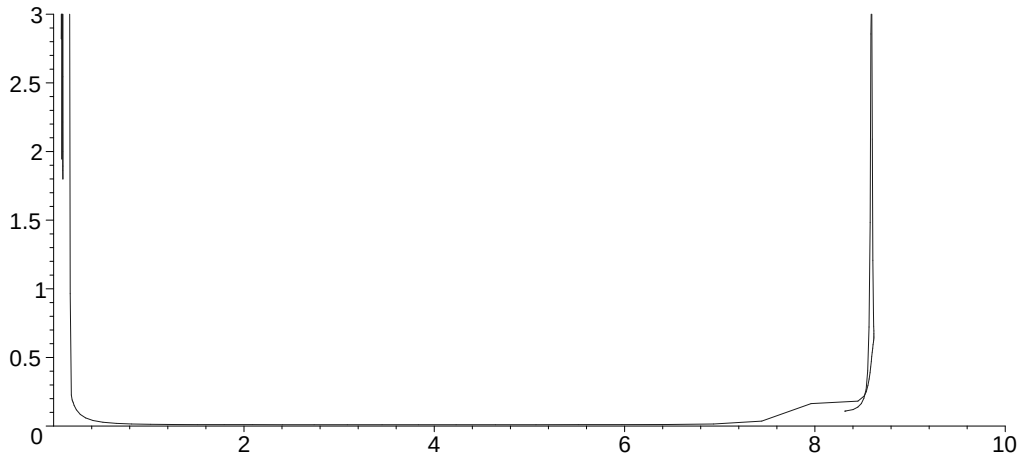
```
> data1 := read_bd("vdp", [2]):
  data2 := NULL:
  for i from 1 by 1 to nops(data1[1]) do
    data2 := data2, [data1[1][i][1],
  evalf(data1[1][i][4]/data1[1][i][5])]:
  od:
  for i from 1 by 1 to nops(data1[2]) do
```

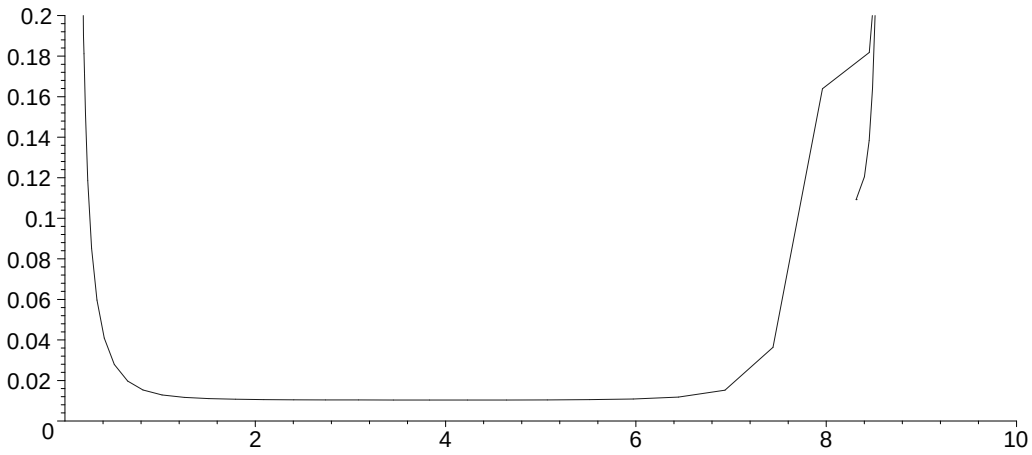
```
  data2 := [data1[2][i][1],
evalf(data1[2][i][4]/data1[2][i][5])], data2:
od:
pl := plot([[data2]], color=blue):
pl := pl, plot([
    [[0,1/1], [10,1/1]],

    [[0,5/4], [10,5/4]],
    [[0,5/3], [10,5/3]],
    [[0,5/2], [10,5/2]],

    [[0,4/3], [10,4/3]],

    [[0,3/2], [10,3/2]],

    [[0,2/1], [10,2/1]],

    [[0,3/1], [10,3/1]]
  ], color=red):
display(pl);
```



A plot of the estimated error of the torus solution. The error is plottet over d (delta). The two areas of the strong 1:1 and 1:3 resonances are clearly visible.

```
> data2 := select_bd_cols(data1, 1, 3):
  plot(data2, 0..10, 0..3, color=blue);
  plot(data2, 0..10, 0..0.2, color=blue);
```


3
2.5
2
1.5
1
0.5
0
2
4
6
8
10


0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
2
4
6
8
10

[>

[>